

PVTM User's Guide (4.7.0.28)

Contents

Chapter 1. Introduction to PVTM.....	8
What is the logic behind PVTM?	8
How much delay in a single server?	9
Threads vs Processes	10
In-house vs Third Party apps	10
Chapter 2. Architecture	11
Deployment Example	12
Security	12
Chapter 3. Requirements.....	14
Chapter 4. Getting Started.....	15
Starting PVTM Simulator and GUI	15
Updating the license.....	16
Running PVTM Agent.....	16
Linux PVTM Agent instructions	16
Linux PVTM Agent instructions with Azul Zing	17
Linux PVTM Agent instructions with Open Onload	17
Linux PVTM Agent instructions with Azul Zing and Open Onload	17
Linux PVTM Agent instructions with TREP and Open Onload	17
Linux PVTM Agent instructions with UMP Stores	18
Linux PVTM Agent instructions with UMP Stores and Open Onload.....	18
Windows PVTM Agent instructions:	18
PVTM GUI Instructions:	19
Elastic Search Notes.....	19
Chapter 5. PVTM Agent	20
Data Collection	20
Target Software Model.....	20
SET_APPS	20
SET_APPLINKS	22
Target Hardware Model.....	23
SET_COMPUTER.....	23
Score Analysis.....	24
GET_SCORE	25
Thread Pinning	25

STOP_SOLUTION request.....	25
STOP_SOLUTION reply	25
RUN_SOLUTION_CACHED request	25
RUN_SOLUTION_CACHED response	27
Customizing When to Run a Solution Request with defaultNeedToRunSolutionCb().....	28
Command and Control.....	29
COMMAND_REQUEST	30
COMMAND_SET_PASSIVE	30
COMMAND_SET_ACTIVE	30
COMMAND_RESET.....	31
COMMAND_GET_CONFIG	31
COMMAND_GET_ACTIVE_STATUS.....	31
COMMAND_SET_CONFIG:SIZE=<size>:<data>	31
COMMAND_FORCE_RUN_SOLUTION_CACHED:SIZE=<size>:<data>.....	31
COMMAND_TRAIN_FUZZY_CACHE:SIZE=<size>:<data>	31
COMMAND_TRAIN_FUZZY_CACHE_STOP	34
COMMAND_QUERY_FUZZY_CACHE:SIZE=<size>:<data>	34
COMMAND_QUERY_FUZZY_CACHE_STOP	35
COMMAND_REPLY.....	35
Command line utilities to send Control Messages.....	36
PVTMAgentRemoteControl.....	36
Terms and Concepts Definitions	38
Command Line	41
Environment Variables	43
LD_PRELOAD.....	43
PVTM_AGENT_CONFIG_FILE_NAME	43
PVTM_AGENT_FILENAME_FORMAT	43
PVTM_AGENT_FORCE_MUX_BUSY_POLL	44
PVTM_AGENT_FORCE_MUX_BUSY_POLL_FILTER.....	44
PVTM_AGENT_FORCE_SO_BUSY_POLL.....	44
PVTM_AGENT_FORCE_SO_BUSY_POLL_FILTER	45
PVTM_AGENT_SHMEM	45
PVTM_DISABLE_HUGE_PAGES	45
PVTM_DISABLE_NUMACTL_LOCAL.....	45
PVTM_ENABLE_SELF_PINNING	45
PVTM_PID_FILE_DIR.....	46
pvtm.cfg Configuration File	46

BNF Grammar	46
Configuration Values	47
pvtm_agent_aggressive_run_solution_if_score_too_high	47
pvtm_agent_app_config_override	47
pvtm_agent_app_link_file_delta_threshold	48
pvtm_agent_app_link_file_delta_threshold_bypass_pre_filter	48
pvtm_agent_app_link_inclusive_regex_filter	48
pvtm_agent_app_link_inet6_socket_delta_threshold	48
pvtm_agent_app_link_inet6_socket_delta_threshold_bypass_pre_filter	48
pvtm_agent_app_link_inet_socket_delta_threshold	48
pvtm_agent_app_link_inet_socket_delta_threshold_bypass_pre_filter	49
pvtm_agent_app_link_lock_delta_threshold	49
pvtm_agent_app_link_lock_delta_threshold_bypass_pre_filter	49
pvtm_agent_app_link_override	49
pvtm_agent_app_link_pair_delta_pcnt_threshold	50
pvtm_agent_app_link_poll_with_thread_info	50
pvtm_agent_app_link_unix_socket_delta_threshold	50
pvtm_agent_app_link_unix_socket_delta_threshold_bypass_pre_filter	51
pvtm_agent_app_link_use_preload	51
pvtm_agent_app_link_use_preload_linux_inject	51
pvtm_agent_app_thread_perf_counter_events_csv	51
pvtm_agent_assume_all_threads_talk	52
pvtm_agent_chrt_rr_matching_processes	52
pvtm_agent_context_switch_cost	52
pvtm_agent_cpu_info_json	53
pvtm_agent_cpu_max_total_pcnt_auto_passive	53
pvtm_agent_cpu_scaling_factor	53
pvtm_agent_cpu_single_core_pcnt_auto_passive	54
pvtm_agent_cpu_state_linux	54
pvtm_agent_cpu_state_linux_percentage_threshold	54
pvtm_agent_delta_cpu_util_threshold	55
pvtm_agent_delta_score_threshold_percentage	55
pvtm_agent_delta_score_threshold_positive_trigger_only	55
pvtm_agent_delta_time_to_delete_old_files_ms	55
pvtm_agent_dynamic_config_file_name	56
pvtm_agent_fake_zero_percentage_threads_last_core	56
pvtm_agent_force_run_solution_request_if_new_threads_start	56

pvtm_agent_ignore_inactive_irqs	56
pvtm_agent_ignore_zero_percentage_threads	56
pvtm_agent_inflate_cpu_util_threshold	56
pvtm_agent_invalid_solution_counter_threshold	57
pvtm_agent_irq_auto_filter_regex_pattern	57
pvtm_agent_irq_auto_force_allowed_cores	57
pvtm_agent_irq_auto_max_total_cpu_util	57
pvtm_agent_irq_files	57
pvtm_agent_irq_route_table_symbols_to_cut	59
pvtm_agent_junk_cores	59
pvtm_agent_junk_cores_json_array	59
pvtm_agent_junk_cores_allow_threads	60
pvtm_agent_junk_cores_sanity_check	60
pvtm_agent_link_proto_weights_un_i4_i6_ld_pf_lk_sm_mins_json_array	60
pvtm_agent_link_proto_weights_un_i4_i6_ld_pf_lk_sm_maxs_json_array	60
pvtm_agent_lock_stat_countdown_to_stop_monitoring	61
pvtm_agent_log_enable_syslog	61
pvtm_agent_log_level	61
pvtm_agent_migrate_memory_when_pinning	61
pvtm_agent_min_score_apply_pinning_threshold	61
pvtm_agent_move_non_match_pids_to_junk_cores	62
pvtm_agent_norm_apps_send_current_cores	62
pvtm_agent_num_times_no_sigificant_changes_force_send	62
pvtm_agent_passive_pinning_mode	62
pvtm_agent_passive_pinning_mode_send_requests	62
pvtm_agent_premium_junk_cores	63
pvtm_agent_process_conn_stats	63
pvtm_agent_process_lock_stats	63
pvtm_agent_regex_pattern	63
pvtm_agent_regex_pattern_results_delimiter	64
pvtm_agent_run_config_apply_pinning_per_thread	64
pvtm_agent_run_config_apply_pinning_per_thread_merge_last_n_solutions	64
pvtm_agent_run_config_apply_pinning_per_thread_merge_last_n_solutions_cpu_threshold	65
pvtm_agent_run_config_apply_pinning_whole_process	65
pvtm_agent_run_solution_cache_app_thread_pcmt_only	65
pvtm_agent_run_solution_cache_enabled	65
pvtm_agent_run_solution_cache_lru_table_size	66

pvtm_agent_run_solution_command	66
pvtm_agent_run_solution_command_after_new_process_delay.....	68
pvtm_agent_run_solution_new_process_delay_ms.....	68
pvtm_agent_same_core_cost.....	68
pvtm_agent_score_callback_lib_file	68
pvtm_agent_score_callback_function_name	68
pvtm_agent_score_moving_average_backoff_if_too_high	69
pvtm_agent_score_moving_average_period	69
pvtm_agent_score_num_ignore_after_pinning	69
pvtm_agent_search_command_line	69
pvtm_agent_search_env_vars.....	69
pvtm_agent_send_thread_names	70
pvtm_agent_set_chrt_rr_priority	70
pvtm_agent_store_last_core_on_command_force_run_solution_cached	70
pvtm_agent_timer_poll_cfg_file_interval_ms	70
pvtm_agent_timer_poll_command_req_interval_ms	70
pvtm_agent_timer_poll_proc_interval_ms	71
pvtm_agent_timer_poll_thread_interval_ms	71
pvtm_agent_unsolicited_run_threshold	71
pvtm_agent_wait_before_sending_run_solution_ms	71
pvtm_agent_wait_outstanding_pinning_request.....	71
pvtm_agent_web_socket_interval_us.....	71
pvtm_agent_web_socket_max_payload_bytes	72
Chapter 6. PVTM Simulator	73
PVTM Score Calculation	73
Context Switches.....	73
Inter-thread Communication Cost.....	75
Thread Execution Layout Simulation	76
Initial Placement.....	76
Score Improvement	76
Historical Repository	77
PVTM Elastic Search Schema.....	77
"RUN_SOLUTION_CHECK_CACHED".....	77
"RUN_SOLUTION_CHECK_CACHED_REQUEST".....	77
"SET_COMPUTER"	78
"SET_APPLINKS".....	78
"SET_APPS".....	78

"RUN_SOLUTION_STATUS"	78
"EVENT"	79
"CONFIG"	79
PVTM Elastic Search Queries	79
Dumping the current schema:	79
Find the Index and Node Shards.....	80
Find all the simulation results between two Timestamps	81
PVTM Elastic Search Configuration	82
Command Line	82
Chapter 7. PVTM GUI	85
Login Screen.....	85
Main Canvas.....	86
Top Toolbar.....	86
Navigation Buttons	87
The Release notes Splash Screen	87
The Time series Graph Window	88
The Time series Toolbar	88
The Time Series Canvas	90
The Preview Area.....	93
The Data Analysis Preview Area.....	94
Summary Tab.....	95
Events Tab	95
Threads Tab	96
Select Thread Data to be Plotted.....	97
Thread Cores Button	98
The Configuration Preview Area.....	99
The Options Grid Tab.....	99
The Raw Text Editor Tab.....	100
The Configuration Preview Toolbar.....	100
The Offline Simulator Window	101
Offline Simulator Window Parts	101
Thread Pinning Layout Toolbar	101
Hardware and Software Components Tree	102
Thread Pinning Layout Tabs	103
Hardware Tab.....	103
Thread Layout Tab	105
The Solution Tab.....	107

The App Links Tab.....	108
The Remote Control Window	109
The Configure Menu	109
The Configuration Preview	110
The Design Components Window	111
The Hardware and Software Components Tree.....	111
Component JSON Editor	112
Preview Canvas	112
Changing the Password	113
Updating the License	113
Command Line	114

Chapter 1. Introduction to PVTM

PVTM is an acronym for Pontus Vision Thread Manager. PVTM accelerates software by optimizing the execution layout of threads for the server where they are currently running. PVTM improves the performance of a variety of applications, ranging from latency in financial services Foreign Exchange pricing systems, to throughput of batch jobs and Database-centric workloads.

What is the logic behind PVTM?

Modern operating systems (OSs) on modern NUMA (non-uniform memory access) servers are not very good at managing threads for performance-sensitive apps. OSs typically balance the load across various cores rather than to focus on application performance. When dealing with performance-sensitive applications, balancing the load across various cores causes application latency to increase significantly.

As an example, in Figure 1 below, there are two Four-CPU servers running the same workload. This hypothetical example shows a pipeline with seven steps. The OS on the left server distributes the red threads across all CPUs in a round robin fashion; by doing this, the distances for data movement are much greater. Therefore, the time to move data between the threads is several times greater than in the server on the right. For many applications, constraining the threads to fewer CPUs can significantly increase performance.

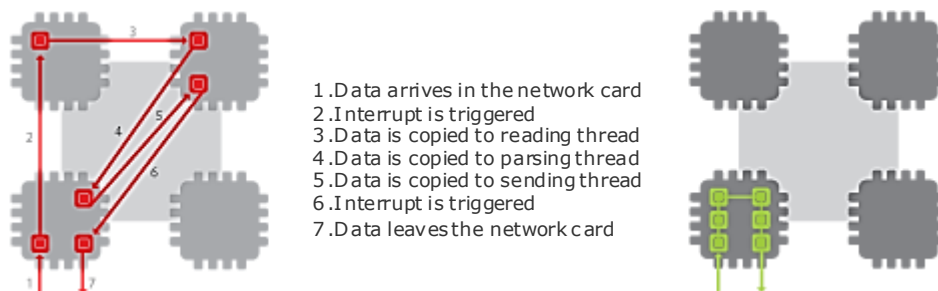


Figure 1 - Scheduler behaviour with and without PVTM

PVTM's powerful simulator finds the best thread pinning layout for a given application. PVTM's patent pending technology allows users to find the optimal thread allocation within seconds. This saves weeks or months of empirical test cycles to find optimized thread allocations. The simulation capabilities also help users determine whether a workload is suitable for new hardware models. Users can collect the software data from a given environment, and simulate what would happen when using different types of server.

PVTM improves the probability that software threads will use less time to communicate with each other. PVTM places threads that have high levels of communication as close to each other as possible whilst at the same time avoiding context switches. To achieve this, PVTM captures the following information about the system:

- Firstly, PVTM analyses the target hardware layout figuring out which cores are closest to each other.

- Secondly, it analyses the communication patterns between threads including which threads use sockets, shared memory and locks to talk to each other.
- Thirdly, PVTM captures the CPU utilization and current location of each thread, as well as optionally the position of I/O devices, such as disks and network cards.

PVTM then sends all this information to a simulator that applies a performance score to the system. The score penalizes threads that have strong communication links to each other, but that are located in cores that have long relative distances to each other. The score also penalizes context switches by avoiding moving active threads to the same core as other active threads as much as possible. The simulator is then capable of running millions of what-if analysis to determine a thread execution layout that improves the score as much as possible.

How much delay in a single server?

Most people think that the delays inside of a single server are negligible; however, the cumulative effect of these delays is very large. The speed at which threads communicate with each other can be over one hundred (100) times slower depending on which cores they are running in a single server. The relative distance between the CPU cores and their memory access affects how quickly data can move between threads.

The following DISTANCE RATIO TABLE shows rough relative speeds of sending data between threads measured on a four CPU server. This table reflects the access speed of memory in different cores, as well as the amount of time that it takes to re-fetch data from main memory if not cached. Access speed is relatively fast when the CPU accesses data stored in level 1, level 2 and level 3 caches. The access speed is slower when the CPU accesses local memory, and even slower when the CPU access remote memory from a neighbouring CPU.

DISTANCE RATIO TABLE

Level 1 Cache	Level 2 Cache	Level 3 Cache	Local Memory	Remote Memory	No-Cross Bridge Memory
1	2	6	10	83	113

It is important to notice that these speeds can vary dramatically depending on the thread behaviour. As such, applications will seldom always run at the fastest speed or at the lowest speed. As the Intel Haswell stats below show, the time to access memory can vary from 4 cycles to access L1 to over 45 cycles plus around 57ns to access main memory. If a thread never moved, and always accessed memory that was within the TLB and in L1 cache, it would use around 4-5 cycles to access the data. In contrast, a thread constantly being moved across cores (in the same CPU) would be missing L1/L2 cached data, having to go to L3 or main memory and having its TLB trashed would use 36 + 9 + 57ns to access the data. In reality, threads will never behave as perfectly as the 4-5 cycles 100% of the time; however, when threads are not pinned, they tend to be constantly in the misbehaved state. By pinning threads in place, PVTM is trying to increase the chances of the good 4-5 cycle behaviour.

Here are some stats for Intel's Haswell processor:

- L1 Data Cache Latency = 4 cycles for simple access via pointer
- L1 Data Cache Latency = 5 cycles for access with complex address calculation (`size_t n, *p; n = p[n]`).

- L2 Cache Latency = 12 cycles
- L3 Cache Latency = 36 cycles
- RAM Latency = 36 cycles + ~ 57 ns
- L1 Data TLB - miss penalty of 1-8 Cycles , with 4x1Gb/ 32x2Mb / 64x4Kb entries
- L2 Data TLB - miss penalty of 9 -22 cycles with 1024 2Mb / 1024 x 4Kb entries

Threads vs Processes

Note that up to this point, we have not mentioned processes, but rather threads. It is easy to confuse the two terms, process and thread. As the name implies, PVTM works at a thread level, and largely does not really care much about the process boundaries. **PVTM can equally optimize single-threaded processes, or multi-threaded processes, or any combinations thereof.**

In most modern operating systems, the main distinction between a few single-threaded processes and a multi-threaded process is their rights to access memory. Threads within a single process can access the same memory area directly, whereas threads that reside in different processes have to make special calls to use shared memory. Many single-threaded applications use shared memory to communicate; similarly, many multi-threaded applications use other inter-process communication (IPC) mechanisms such as sockets or named pipes between their threads. The reasons and merits of using single-threaded architectures vs multi-threaded architectures and their IPC mechanisms vary dramatically, and it is beyond the scope of this document to discuss them.

PVTM's model makes no distinction between two threads that belong to the same process and decide to use futex locks to lock a shared memory area, and two threads that belong to different processes, and use futex locks to lock a shared memory area. As long as there is a strong communication pattern between the threads, we do not really care to which process they belong.

In-house vs Third Party apps

PVTM works with both in-house and third party apps, brand new apps and legacy apps equally. PVTM is also agnostic to which computer language is used. PVTM captures low-level system calls to determine which threads communicate with each other. It behaves like a high-performance profiler that captures basic system calls with little overhead to the system. Because these calls are at quite a low-level, PVTM can easily figure out inter-thread patterns between threads in languages as diverse as C, C++, Java, R, Perl, C#, Python, and any other higher level language, as long as the applications are dynamically-linked against the system calls.

Chapter 2. Architecture

As seen in Figure 2 below, PONTUS VISION Thread Manager (PVTM) has 3 main components:

- 1) PVTM Agent – (pvtm-agent) – A lightweight single threaded agent that collects information about the hardware, and discovers the data communication patterns between threads. PVTM Agent is written in C, and usually uses < 1% CPU to capture its data. To aid PVTM Agent discover the thread pinning strategies, the following components may also be used:
 - a) libpvtm-agent-preload.so – On Linux, a library that can help the PVTM Agent discover communication patterns between threads. This can be injected in existing applications without recompiling them by using the LD_PRELOAD environment variable.
 - b) pvtm-agent.jar – an optional java agent file that exposes the names of Java threads to the operating system. To use this, you need to change your JVM command line to add the -javaagent:<path to the pvtm-agent.jar file>.
 - c) pvtm-agent-preload-windows.dll – On Windows, a library that PVTM Agent discovers communication patterns between threads. PVTM Agent automatically connects to the applications that need to be monitored using this DLL along with remote debugging techniques
- 2) PVTM Simulator / Thread Manager – (run-threadmgr.sh) – a Java 7 standalone simulation engine that receives TCP/IP connections from PVTM Agent, and can take the hardware information, as well as the data communication patterns between the threads to produce an optimal layout of software threads on the hardware cores.
- 3) PVTM GUI Server (run-gui.sh) – a self-contained server that hosts a browser-based graphical user interface. The GUI enables users to visualize the layout of the threads, and produce scripts for static thread pinning configurations. This can be used in environments where PVTM Agent is not allowed to run.

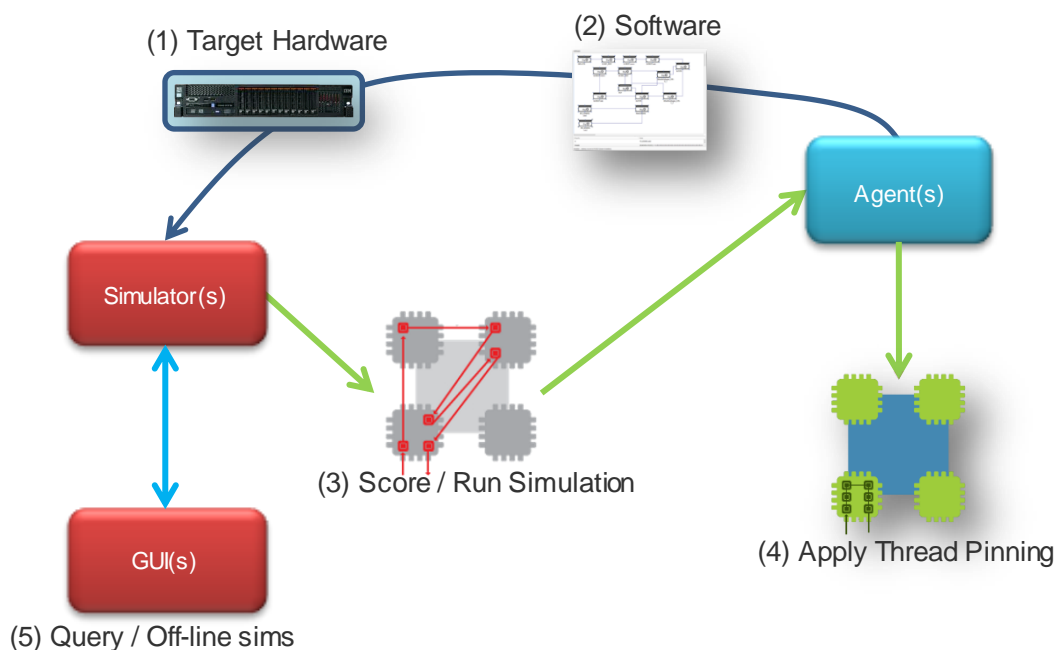


Figure 2 - PVTM Component Architecture

Deployment Example

Figure 3 below shows a small deployment of thread manager. The ratio of PVTM Agents to PVTM Simulators increases over time. Each simulation takes around 10 to 30 seconds to execute, but the PVTM Simulator is an Elastic Search node that stores previous simulation results. As the number of previous simulation results increases, the PVTM Simulator spends less and less time running calculations and the amount of CPU required decreases drastically.

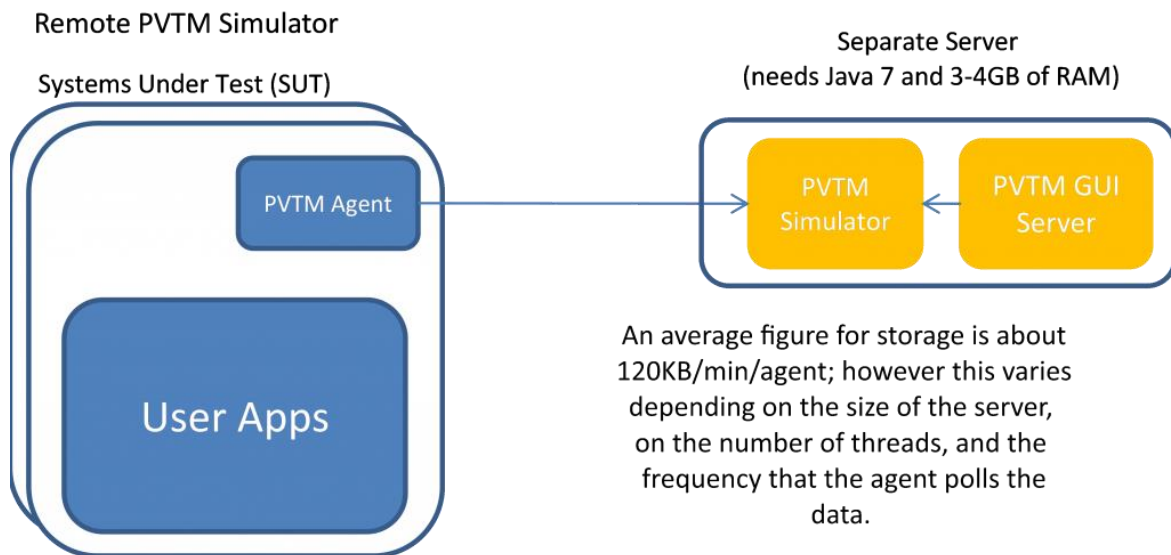


Figure 3 - Deployment Example

We have seen installations where users have a 100 to 1 ratio of agents to simulators, and after the learning curve, up to 1000 to 1 ratio; however, that figure is highly dependent on the complexity of the system, how much the workloads vary, and how often the PVTM Agent is configured to poll information. The PVTM Simulators can scale horizontally, as long as there is TCP/IP connectivity between Simulator instances, they will start forming an Elastic Search cluster and data will be sharded across the nodes as they scale.

Security

PVTM comes with a default key store that contains self-signed certificates. These will generate warnings in browsers stating that the web site is unsafe. It is recommended that the user provides his or her own store with certificates signed by a Certificate of Authority recognized by the corporate browsers. The creation of a certificate store is beyond the scope of this document; however, the user may need to modify some of the following settings in the run-gui.[bat/sh] file used to start the PVTM GUI server.

Tip: All the -D command line configurations may also be put inside of a properties file, and passed in as the first line argument to the PontusVision server; this is far safer than passing them on the command line, especially if the keystore and truststore passwords are kept in clear text. Passing these on the command line easily allows anybody in the machine to run commands like ps, and get the list of arguments.

Here's a sample format for the property file:

```
pontusvision.truststore=/opt/pontusvision/.keystore_pontusvision
```

pontusvision.web_data_keystore_alias=localhost
 pontusvision.keystore=/opt/pontusvision/.keystore_pontusvision

Parameter	Default	Description
-Dpontusvision.web_data_keystore_alias	localhost	The certificate alias used by the web server to encrypt the https connections.
-Dpontusvision.keystore	<home>/.keystore_pontusvision	Full path to the Keystore file name
-Dpontusvision.truststore	Same as pontusvision.keystore	Full path to the TrustStore file name
-Dpontusvision.keystore_password	pa55word	Clear-text passwords; please ensure that this configuration is kept secure, ideally in a property file (see tip above)
-Dpontusvision.keymanager_password	pa55word	
-Dpontusvision.truststore_password	pa55word	

Chapter 3. Requirements

PVTM has specific requirements for each of its components. The PVTM Simulator and GUI can be deployed on any Operating Systems that support Java 7 or later. The PVTM Agent is written in C, and is currently supported on Linux, Windows, as well as an experimental version for AIX 7.1. Here are the technical requirements for each component:

- PVTM Simulator & PVTM GUI Server – Java 7 or later
- PVTM Agent (Linux) – glibc 2.4 or later – root access is desirable, but not strictly needed
- PVTM Agent (Windows) – Windows 2003 and newer – with admin rights, and if using virus checkers, ensure that PVTM Agent is added to the exception list
- PVTM GUI – Firefox or Chrome web browsers
- PVTM Agent hardware – bare metal, or non-overcommitted VM (1:1 ratio of dedicated physical to virtual cores) with 8 + cores
- PVTM Simulator and GUI hardware – 4 cores, 4 GB of memory, 10 GB of storage.
- Network connectivity: tcp connections to be established using the same arrow directions as shown in Figure 3.

End users perform their operations on workstations using a Web browser. The following browsers are supported:

- Recommended Google Chrome (desktop version 24 or higher)
- Mozilla Firefox (desktop version 17.0 or higher)
- Microsoft IE11 or above
- Microsoft IE7 or IE8 – only supported with the following plug-in:
<https://developers.google.com/chrome/chrome-frame/>

Note: if your firewall is blocking it, you can also get it from here; once you download the file, rename it as a .zip file, and use the password 'pontus' to extract the chrome-frame offline installer

Chapter 4. Getting Started

To get started quickly, these are the install instructions for Pontus Vision Thread Manager. As seen in the Architecture Chapter, the product is broken into 3 main components:

- 1) PVTM Simulator - A thread manager simulator, which should run on a separate server than the one trying to be optimized
- 2) PVTM GUI - a server that hosts a web-based interface, which may sit either in the same server, or separately to the thread manager simulator
- 3) PVTM Agent - an agent collects performance information about applications, and sends them to the thread manager simulator. The agent will usually run on its own, inside the server that needs to be optimized.

The key requirements are java 7 for the thread manager and GUI servers, and Firefox / Chrome for the user interface. The agent can run on Linux (with glibc 2.4 or above), or Windows. It is recommended that where possible, the agent runs as root on Linux (though that is not required for capturing information), and runs as administrator on Windows.

Here is a Link to the latest version:

<http://www.pontusvision.com/wp-content/uploads/2014/01/pontusvision-4.7.0-installer.jar>

If you do not have a valid licence key, please contact us at sales@pontusnetworks.com, and we will send a new license file, and corresponding key with the following format:

O=<company>,OU=<environment or dept>,CN=<timeframe or person>

Starting PVTM Simulator and GUI

Here are the instructions to run the PVTM Simulator and PVTM GUI:

- 1) choose a server (the PVTMServer) that has connectivity to the server that you want to optimize to run the thread manager server, download the jar file above, and run the following command:

```
java -jar pontusvision-4.7.0-installer.jar -console
```

- 2) use all the default options for now, and when prompted for a license key, please enter the following string:

```
O=<company>,OU=<environment or dept>,CN=<timeframe or person>
```

- 3) once the install finishes, replace the license.lic file in the installation area (by default ~/pontus-vision/4.7.0/license.lic) with the one attached in the e-mail from your pre-sales engineer.

- 4a) Linux instructions - to start the main components, run the following commands:

```
cd ~/pontus-vision/4.7.0/  
./run-threadmgr.sh  
./run-gui.sh
```

- 4b) Windows instructions - to start the main components, perform the following actions:

In the Start Menu, type Pontus, and start the following processes

Start PontusVision Thread Manager
Start PontusVision Server

Updating the license

The license file can be updated from the PVTM GUI (see [Updating the License](#)), or directly by overriding the current license.lic file in the file system. Note that during evaluations, the license key information (e.g. O=<company>,OU=<environment or dept>,CN=<timeframe or person>) may occasionally change. If it does, then this information must be updated in the run-gui.(sh|bat) and run-threadmgr.(sh|bat) files by changing the following line:
-Dpontusvision.license.holder="O=xxxxxx, OU=yyyy, CN=zzzz "

Running PVTM Agent

Once PVTM Simulator and PVTM GUI are running, Here's what you need to run in the target machines where you want the agent to profile the apps:

Linux PVTM Agent instructions

1) cd to the <install directory>/4.7.0/linux, and edit the pvtm.cfg file, changing the pvtm_agent_regex_pattern option with a regular expression that matches a list of applications you want to monitor (e.g. set it to pvtm_agent_regex_pattern="java" to match all the java instances in the server. Note that you can specify a list of processes by delimiting them with a | (e.g. pvtm_agent_regex_pattern="java|apache" will pick up both java and apache instances.

2) tar / zip up the following files from the PVTMServer installation directory:

```
cd ~/pontus-vision/4.7.0/  
tar cvf pvtm-agent.tar linux
```

3) copy the pvtm-agent.tar file to the machines that need to be monitored, and untar them:

```
mkdir -p /opt/pontus-vision/4.7.0  
cd /opt/pontus-vision/4.7.0  
tar xvf pvtm-agent.tar
```

4) run the following:

```
./pvtm-agent --url=<PVTMServer Address>
```

e.g.:

```
./pvtm-agent --url=192.168.51.2
```

5) In order to introspect applications a bit further (e.g. to find whether threads are sharing the same shared memory areas, or futex locks), we also have a shim layer that can unobtrusively intercept system calls you are making. To enable this in your apps, you can set the LD_PRELOAD environment variable, setting it to the libpvtm-agent-preload.so file.

e.g.:

```
export LD_PRELOAD=/opt/pontus-vision/4.7.0/linux/'$LIB'/libpvtm-agent-preload.so  
<run your app normally>
```

NOTE: The '\$LIB' is a placeholder replaced by the ld.so loader to either lib or lib64 depending on whether the app is 32 or 64 bits; also, if you need to put the LD_PRELOAD value in double quotes, then the syntax is as follows:

```
export LD_PRELOAD="/opt/pontus-vision/4.7.0/linux/\$LIB/libpvtm-agent-preload.so"
```


6) To help add custom relationships between threads in Java, we also ship a java agent that exposes the Java thread names to the operating system. To enable this feature, add the following line to your Oracle Hotspot JVM 7 or above:

```
-javaagent: /opt/pontus-vision/4.7.0/linux/pvtm-agent.jar
```

Linux PVTM Agent instructions with Azul Zing

Azul Zing is a high-performance Java Virtual Machine that never pauses for garbage collection. We have created a special version of the LD_PRELOAD library that works with Azul's Zing JVM's memory management implementation. If using Zing, you must use this library instead of the standard one, as otherwise, the application will likely deadlock. The zing library is currently only shipped in 64-bit mode. To use it, simply set the LD_PRELOAD environment variable as follows:

```
export LD_PRELOAD =/opt/pontus-vision/4.7.0/linux/libpvtm-agent-preload-  
zing.so
```

Linux PVTM Agent instructions with Open Onload

Open Onload helps users accelerate network traffic sent through Solarflare's network cards by bypassing the kernel. To use Open Onload with the normal PVTM PRELOAD, do the following:

```
unset LD_PRELOAD  
export ONLOAD_PRELOAD="/opt/pontus-vision/4.7.0/linux/libpvtm-agent-  
preload.so /lib64/libonload.so"
```

Lastly, prefix your application with `onload -profile=<latency|safe|custom>` followed by the normal command line.

Note that the PVTM library must always come before the `libonload.so` file in the `ONLOAD_PRELOAD` environment variable, and that the two files are separated by a space.

Linux PVTM Agent instructions with Azul Zing and Open Onload

The instructions to run Azul Zing with Open Onload are quite similar to the Linux PVTM Agent instructions with Open Onload, with the key difference being the name of the Library in the `ONLOAD_PRELOAD` environment variable:

```
unset LD_PRELOAD  
export ONLOAD_PRELOAD="/opt/pontus-vision/4.7.0/linux/libpvtm-agent-preload-  
zing.so /lib64/libonload.so"
```

Lastly, prefix your application with `onload -profile=<latency|safe|custom>` followed by the normal command line.

Note that the PVTM library must always come before the `libonload.so` file in the `ONLOAD_PRELOAD` environment variable, and that the two files are separated by a space.

Linux PVTM Agent instructions with TREP and Open Onload

TREP is a leading market data distribution system from Thomson Reuters. Starting with version 4.7.0.7, we have released an experimental feature that allows Thomson Reuters's TREP to run with

onload. Traditionally, TREP applications do not work with Open Onload; however, we have created a patch that enables Open Onload to be used with TREP applications by doing the following:

```
unset LD_PRELOAD
export ONLOAD_PRELOAD="/opt/pontus-vision/4.7.0/linux/libpvtm-agent-
preload.so /lib64/libonload.so"
export PVTM_AGENT_ONLOAD_REUTERS_OVERRIDE=1
```

Lastly, prefix your application with `onload -profile=<latency|safe|custom>` followed by the normal command line (e.g. `onload --profile=latency numactl -N 1 --preferred=1 ./ads -nodaemon`)

Note that the PVTM library must always come before the `libonload.so` file in the `ONLOAD_PRELOAD` environment variable, and that the two files are separated by a space.

Linux PVTM Agent instructions with UMP Stores

UMP Stores are a part of Informatica's Ultra Messaging (AKA 29west) low latency middleware suite. Here are the instructions to run UMP Stores with PVTM Agent:

```
export LD_PRELOAD=/opt/pontus-vision/4.7.0/linux/'$LIB'/libpvtm-agent-
preload.so
export PVTM_DISABLE_HUGE_PAGES=1
./umpstored <command line args>
```

Linux PVTM Agent instructions with UMP Stores and Open Onload

UMP Stores are a part of Informatica's Ultra Messaging (AKA 29west) low latency middleware suite. Here are the instructions to run Ump Stores with Solarflare's Open Onload:

```
unset LD_PRELOAD
export ONLOAD_PRELOAD="/opt/pontus-vision/4.7.0/linux/libpvtm-agent-
preload.so /lib64/libonload.so"
export PVTM_DISABLE_HUGE_PAGES=1
onload -profile=latency ./umpstored <command line args>
```

Note that the PVTM library must always come before the `libonload.so` file in the `ONLOAD_PRELOAD` environment variable, and that the two files are separated by a space.

Windows PVTM Agent instructions:

1) Create a batch file (e.g. `run-pvtm-agent.bat`) inside the windows folder in the PVTMServer installation directory with the following:

```
cd C:\Pontus\pontus-vision\4.7.0\windows
sudo pvtm-agent-win.exe -u <PVTMServer Address>
```

2) zip up all the files in the windows folder inside the PVTMServer installation directory

3) copy the zip file from (2) to the machines that need to be monitored, and unzip the file in the `C:\Pontus\pontus-vision\4.7.0\windows` directory

4) run the agent (note that you'll need an account with admin rights):

```
run-pvtm-agent.bat
```

PVTM GUI Instructions:

Users should then be able to point a browser to `https://<PVTMServer Address>:8443` and login as `root/pa55word`.

Once in, click on the Self Tuning button, and choose your server from the drop down list. Once you find a timeline in the graph that you're interested in, just click on that point in the map, and it should bring up the code you need.

Elastic Search Notes

PVTM uses elastic search to store some of our configuration; here are the instructions if you need to change the default port 9200 to a different one (e.g. 9036), or to run the GUI web server (`run-gui.sh`) on campus, pointing to the thread manager (`run-threadmgr.sh`) in co-lo:

a) Create a file called `pvtm.yml` in the thread manager simulator server's (in colo) install dir, with the following line:

```
transport.tcp.port: 9036
```

b) Add the following option to the `run-threadmgr.sh` file in the thread manager simulator server's (in colo):

```
-Dpv.elasticsearch.node.config.url=file://<path to the pvtm.yml file from (a)>
```

c) Add the following options to the `run-gui.sh` file in the GUI Server (campus):

```
-Dpv.elasticsearch.transportclient.host=<IPaddress of enyrs040011> \  
-Dpv.elasticsearch.transportclient.port=9036
```

Chapter 5. PVTM Agent

PVTM Agent runs in each server that needs to be optimized. It performs four main tasks:

- 1) Data Collection – PVTM Agent collects information about the hardware, and builds a cost matrix showing the relative distances between each of the cores in the server; PVTM Agent is also configured to capture information about selective processes in the server that match a regular expression filter. Lastly, PVTM Agent also collects information about network and disk device interrupts, including in which cores the interrupts are running.
- 2) Score Analysis – PVTM Agent keeps a moving average of previous scores, and decides whether to request a solution to be solved by the simulator. The default criterion is to check whether the latest score is within 5% of the moving average; if it is greater, a simulation request is sent to the simulator.
- 3) Thread Pinning – PVTM Agent is also responsible for applying the thread execution layout from a simulation. It moves the threads from their current position to the selected position.
- 4) Command and Control – PVTM Agent can be remotely controlled for actions such as making it passive, active, getting the current status, and even setting the configuration file contents.

PVTM Agent uses ascii text messages sent over web sockets to communicate with the Simulator. Where the content of the messages requires more complex structure, a java script object notation (JSON)-like structure is used. The following sections go over the format of the messages sent between PVTM Agent and the simulator, as well as some of the key concepts behind them.

Data Collection

PVTM Agent collects data related to the Target Software Model, and the Target Hardware Model to provide PVTM Simulator with enough information to calculate scores and run simulations. The following sections show the type of data collected and the message formats used to send it to PVTM Simulator.

Target Software Model

The Target Software Model uses two key JSON-style message types:

SET_APPS

The SET_APPS message has information about all the applications and their threads, including the CPU utilization of each thread, as well as the PIDS, TIDS, and optional information about threads. The SET_APPS message has an array of JSON-style objects with the following fields:

- 1) type – the type of application (either 'pontusdraw2d.application' to indicate a normal application, or 'pontusdraw2d.nicdriver' to indicate a network interrupt request handler)
- 2) a list of properties, including the following values:
 - a. id – the process name as matched by the PVTM Agent's regex, followed by a process ID

- b. threads – a string with a JSON array inside representing each of the threads' percentage CPU utilizations
- c. pvtmThreadIds – an optional string with a JSON array inside representing the original thread IDs
- d. pvtmCurrentCores – an optional string with a JSON array inside representing the current cores where each of the threads was running
- e. assumeAllThreadsTalk – a Boolean flag that indicates whether the model should artificially create a link between all the threads within the process. Note that this is not typically advised unless PVTM Agent is unable to capture the inter-thread communication patterns.
- f. allowedCores – an optional string with a JSON array inside representing the list of cores where all the threads in the application are allowed to run. The simulator model will treat this as a hard constraint, and will not produce a valid score until the threads are running on the correct cores. Note that some threads that are idle do not obey the thread pinning strategy until they run again, and as such, may erroneously appear in the wrong core in tools like 'top' on Linux. PVTM Agent is able to report these threads in the new core allocation; when the threads run again, they should be scheduled in the new cores.
- g. pvtmPerfCounterEventsCsv - an optional comma-separated string that determines which performance counter events PVTM Agent will send in the pvtmPerfCounterVals option (see pvtm_agent_app_thread_perf_counter_events_csv for more details).
- h. pvtmPerfCounterVals - an optional two-dimensional array that has an array of performance counter values (matching the number of events sent in the pvtmPerfCounterEventsCsv optional string) for each thread. See pvtm_agent_app_thread_perf_counter_events_csv for more details.
- i. threadAllowedCores - an optional two-dimensional array that has an array of allowed cores for each thread in the application. An array with a single value of -1 means the thread can run anywhere. Arrays with one or more numbers greater than or equal to zero will constrain the PVTM Simulator from moving those threads to just the cores listed. If this is not present, the threads can be placed anywhere. Note that the allowedCores option should always be a super-set of the values listed here. If not, the allowedCores for the whole process will trump this option.

Here is a sample message:

```
[
{
  'type': 'pontusdraw2d.application',
  'properties':
  {
    'id': 'test-6440',
    'threads': '[0,0,16,0]',
    'pvtmThreadIds': '[86500,86380,86492,86504]',
    'pvtmCurrentCores': '[1,1,3,3]',
    'assumeAllThreadsTalk': 'false',
    'pvtmPerfCounterEventsCsv': 'L1-dcache-load-misses,L1-dcache-store-
misses,LLC-load-misses,LLC-store-misses,dTLB-load-misses,dTLB-store-misses',
    'pvtmPerfCounterVals': '[[0,0,0,0,0], [10,10,2,3,4,6],
[34,333,32,11,11,33], [23,55,55,21,35,65]]'
    'threadAllowedCores': '[[ -1], [-1], [2,3], [2,3]]'
  }
}
```

```

},
{
  'type': 'pontusdraw2d.nicdriver',
  'properties':
  {
    'id': 'IRQ--110',
    'threads': '[50]',
    'allowedCores': '[0,1]'
    'pvtmThreadIds': '[-110000000]',
    'pvtmCurrentCores': '[1]'
  }
}]

```

SET_APPLINKS

The SET_APPLINKS message has information about all the inter-thread communication patterns, including the type of connection, as well as the volumes of data. PVTM Agent typically captures this information after a shim layer of code (either using LD_PRELOAD on Linux, or a remote debugger on Windows) intercepts the application's calling patterns. This information can also be injected by adding a set of rules in the pvtm.cfg Configuration File.

The SET_APPLINKS message has an array of JSON-like objects with the following fields:

- 1) sourceId – the id (as defined in the SET_APPS message) of the process starting the connection.
- 2) targetId – the id (as defined in the SET_APPS message) of the process ending the connection.
- 3) sTid – the source thread ID (as defined in the SET_APPS message) of the thread starting the connection.
- 4) tTid – the target thread ID (as defined in the SET_APPS message) of the thread ending the connection.
- 5) Type – a string with the following general format:

<Type>_<Conn details>_(<Connection gross weight>), where:

Type is "local_domain_socket", "lock", "inet_socket", "inet6_socket", "pipe_or_file" or "shmm"

Conn Details are the connection details (e.g. for inet_socket, the source and destination ports, source and destination IP addresses, for pipe or file the name of the pipe or file name, for lock the memory address of the lock in use, and for shmm the shared memory address.

Connection gross weight is the non-normalized number of bytes sent & received by the threads for the local_domain_socket, inet_socket, inet6_socket, pipe_or_file types, and the number of times a lock was unlocked for the lock type, and the number of times a shared memory area was created for shmm.

Here's a sample message:

```

[
  {
    'sourceId': 'test-6440',
    'targetId': 'IRQ--110',
    'sTid': '86500',
    'tTid': '-110000000',
    'type': 'inet6_socket_lport=57278,rport=61616_(5555556)'
  },

```

```

    {
      'sourceId': 'test-6440',
      'targetId': 'test-6440',
      'sTid': '86500',
      'tTid': '86380',
      'type': 'lock_ptr=9F6FC0_(80)'
    },
    {
      'sourceId': 'test-6440',
      'targetId': 'test-6440',
      'sTid': '86492',
      'tTid': '86504',
      'type': 'pipe_or_file_EventFD1243_(1234566)'
    }
  ]

```

Target Hardware Model

The Target Hardware Model is uses a single a JSON-like object that describes the server where PVTM Agent is running, as well as a few parameters used by the simulator to apply the system score.

SET_COMPUTER

The SET_COMPUTER message is a JSON-style object that has the following fields:

- 1) Type – currently hardwired to `pontusdraw2d.IBM_3750_m4`
- 2) Properties – a JSON-style object with the following fields:
 - a. Id – an identifier for the server
 - b. ContextSwitchCost – the multiplier that the simulator will use to penalize the score for context switches. The formula used by the simulator is the following:

$$\text{ContextSwitchCost} = \frac{\ln(\text{numThreads}) * \text{CPUUtil} * \text{ContextSwitchCost}}{\text{CpuScalingFactor}}$$

Where:

numThreads is the number of threads currently scheduled in the core,

CPUUtil is the total CPU percentage of the threads currently scheduled in the core, or 0.1 if the total utilization is zero)

ContextSwitchCost is the value set in the SET_COMPUTER message

CPUScalingFactor is either 1, or the value set in the RUN_SOLUTION_CACHED request message

- c. junkCores is a string with a JSON array representing a list of cores that the simulator is not allowed to use in its model; any threads running in a junk core yield an invalid score.
- d. physicalCpuList is a string with a JSON array representing the physical CPU socket numbers of each of the cores in the server
- e. latencyMatrix is a sparse matrix of integers that has the relative costs of sending data from one core to another core. These values are unit-less, and only represent a normalized cost of sending data from one core to another.
- f. UN_I4_I6_LD_PF_LK_SM_MINS/MAXs – These two options allow users to bias the weights of inter-thread connections. They contain the min and max values for the normalized weights of the following protocols:

- un - unknown protocol
- i4 - ipv4 sockets
- i6 - ipv6 sockets
- ld - local domain sockets
- pf - pipes and files
- lk - locks
- sm - shared memory

These arrays will spread the actual values for the protocols (the value in parenthesis of how many unlocks for locks or bytes were sent on sockets between these artificial min and max values. The default values below will make all the unknown protocols with a weight of 1, all the ipv4,v6, and local domain sockets spread between 20 and 30, all the pipes and files between 1 and 5, all the locks at 1, and all the shared memory areas at 10. These will effectively bias the model towards favouring socket traffic over locks, pipes and files, and shared memory.

Here is a sample message describing a 4-core server with a single CPU socket:

```
{
  "type": "pontusdraw2d.IBM_3750_m4",
  "properties":
  {
    "id": "id",
    "contextSwitchCost": "100",
    "junkCores": "[]",
    "physicalCpuList" : "[0,0,0,0]",
    "latencyMatrix": "[[100,1,8,8],
                      [1,100,8,8],
                      [8,8,100,1],
                      [8,8,1,100]]",
    "UN_I4_I6_LD_PF_LK_SM_MINS": "[1,20,20,20,1,1,10]",
    "UN_I4_I6_LD_PF_LK_SM_MAXS": "[1,30,30,30,5,1,10]"
  }
}
```

Score Analysis

In addition to the data collection messages in the previous section, PVTM Agent also sends score requests to the simulator. The score requests are one of the most important activities of PVTM Agent. The scores calculated by the simulator influence when PVTM Agent decides to request, and to apply thread pinning strategies.

The score requests are sent whenever the CPU utilization of any of the threads changes by a configurable percentage threshold, whenever a new thread starts, or PVTM Agent can also be configured to force a score request to be sent periodically regardless of any changes in the environment.

GET_SCORE

The GET_SCORE message only has the same payload as the value set in the PVTM Agent's `pvtm_agent_run_solution_command` option. When the simulator receives this message, it takes the latest state of the system, as defined by the last SET_COMPUTER, SET_APP and SET_APPLINK messages received, and uses that to calculate a score.

The return value is either an ASCII text number representing the score, or an error message, such as "INVALID SOLUTION": followed by a score, or "ERROR". An INVALID SOLUTION message appears whenever the current state of the system has threads running in a core that exceed 100% CPU utilization, or a thread running in a core that is not in its 'allowed cores' list. If PVTM Agent receives an invalid score, it will immediately try to send a request to run a simulation, which is covered in the Thread Pinning section. If PVTM Agent receives a valid score, it calls the `defaultNeedToRunSolutionCb()`, also described in the Thread Pinning section to determine whether a new thread pinning strategy is required.

Thread Pinning

In addition to data collection and thread pinning, thread pinning is the third type of activity that enables PVTM Agent to stop simulations, to run new simulations, and to get simulation results to be applied automatically in the system. The thread pinning activities use four message types:

STOP_SOLUTION request

Whenever a simulation request needs to be sent to the simulator, a STOP_SOLUTION request message is sent first. This ensures that any currently running simulations are stopped before requesting a new one. The body of the STOP_SOLUTION request is currently unused.

STOP_SOLUTION reply

The reply message from a stop request is always either the string "ERROR" or "OK". It is very rare for an ERROR response to be received. If it does arrive, please look at the simulator server logs, and report any exceptions to the PVTM support team.

RUN_SOLUTION_CACHED request

Once a successful STOP_SOLUTION request arrives back at PVTM Agent, a RUN_SOLUTION_CACHED request message is sent. The RUN_SOLUTION_CACHED request message has a normalized request, which strips the process names and process IDs so the results can be more easily cached and re-used across instances. The normalized request (`normReq`) object is fully self-contained, and has three main components:

- 1) `computerJson`, which is similar to the [SET_COMPUTER](#) message,
- 2) `normAppsJson`, which is similar to the [SET_APPS](#) message, except that the process names and thread IDs are replaced with increasing integers starting at 0, and the `pvtmCurrentCores` field is not present

- 3) normAppLinksJson, which is similar to the [SET APPLINKS](#) message, except that the sourceID, targetID, sTid and tTid are replaced with increasing integers starting at 0, matching the values from the normAppsJson.

In addition to the normReq, the RUN_SOLUTION_CACHED request message also has the following fields:

- 1) normReqHash – a 64-byte hash that is used for finding previously saved simulations
- 2) threadLastCores:[1,1,3,3,1] – an array of cores for each of the threads in the model
- 3) 'threadIds':[86500,86380,86492,86504,] – an array of the actual thread IDs
- 4) 'reason' – a string describing why the thread pinning strategy was sent
- 5) 'runSolutionCommand': a string that controls simulator parameters such as how long to run the simulation for, whether or not to randomize the core assignments, and to clear the last core assignments before running a simulation (see the section on pvtm_agent_run_solution_command for more details about the format of this parameter).

Here is an example RUN_SOLUTION_CACHED request message:

```
{
  normReq:
  {
    computerJson:
    {
      "type": "pontusdraw2d.IBM_3750_m4",
      "properties":
      {
        "id": "id",
        "contextSwitchCost": "100",
        "junkCores": "[]",
        "physicalCpuList" : "[0,0,0,0]",
        "latencyMatrix": "[[100,1,8,8],
                          [1,100,8,8],
                          [8,8,100,1],
                          [8,8,1,100]]",
      }
    }
  }, normAppsJson: [
    {
      'type': 'pontusdraw2d.application',
      'properties':
      {
        'id': '0',
        'threads': '[0,0,16,0]',
        'pvtmThreadIds': '[0,1,2,3]',
        'assumeAllThreadsTalk': 'false'
      }
    },
    {
      'type': 'pontusdraw2d.nicdriver',
      'properties':
      {
        'id': '1',
        'threads': '[50]',
        'allowedCores': '[0,1]',
        'pvtmThreadIds': '[4]',
      }
    }
  ]
}
```

```

    }
  ],
  normAppLinksJson:[
    {
      'sourceId':'0',
      'targetId':'1',
      'sTid':'1',
      'tTid':'4',
      'type':'inet6_socket_0_(20)'
    },
    {
      'sourceId': '0',
      'targetId': '0',
      'sTid':'0',
      'tTid':'1',
      'type':'lock_ptr_0_(1)'
    },
    {
      'sourceId':'0',
      'targetId':'0',
      'sTid':'2',
      'tTid':'3',
      'type':'pipe_or_file_EventFD1243_(1)'
    }
  ]
}

```

```

'normReqHash':"0eb068387b46dc5e3a88bb152e538fe2065407eb942332282ad5d801cca20
bb9",
  'threadLastCores':[1,1,3,3,1],
  'threadIds':[ 86500,86380,86492,86504, -110000000],
  'reason':"39.34 Percentage difference was higher than the threshold of
10 New score = 13450; last simple moving avg (sma) = 8978.60; new sma =
8978.60",
  'runSolutionCommand':
    "{ 'maximumSecondsSpend':'30','randomize':'false', 'clearLast':'true',
'cpuScalingFactor': '1'
}"
}

```

RUN_SOLUTION_CACHED response

Once the simulator completes a RUN_SOLUTION_CACHED simulation request, a RUN_SOLUTION_CACHED response message is returned back to PVTM Agent. This message has enough information for PVTM Agent to pin threads to specific cores, and on Linux, the IRQs to specific CPU masks.

The RUN_SOLUTION_CACHED response message has an array of JSON-like objects for each of the processes the following fields:

- Pid – the process ID of the process
- threadIds – a JSON array of thread IDs
- coreIds – a JSON Array of core IDs representing the location of each of the threadIds by order (e.g. in the example message, thread ID 9672 would be pinned to core 0, thread ID 9684 to core 4, and thread ID 9689 to core 4).
- numaZoneIds – a 2-Dimensional JSON Array with the NUMA (Non-uniform memory access) zones where each of the threads is allowed to run. PVTM Agent currently ignores this feature.
- Score – the pre-calculated score for the current thread execution pattern.

```
[{"pid":9672,"threadIds":[9672,9684,9689],"coreIds":[0,4,4],"numaZoneIds":[[0],[0],[0]],"score":"2217"}]
```

Customizing When to Run a Solution Request with defaultNeedToRunSolutionCb()

This function determines both whether a RUN_SOLUTION_CACHED request message should be sent to the simulator, and whether a RUN_SOLUTION_CACHED response should be applied. Users can easily override this function with any custom logic to determine when to execute a thread pinning strategy. See the `pvtm_agent_score_callback_lib_file` section for more details on how to load this into PVTM Agent.

The `defaultNeedToRunSolutionCb()` function returns 1 if a new thread pinning strategy should be requested, or applied, or 0 otherwise

The default implementation uses a moving average of previous scores to determine whether a new thread pinning execution layout is required. Here is the function prototype:

```
static int defaultNeedToRunSolutionCb
(
    long cfgMinScoreApplyPinningThreshold,
    int cfgDeltaScoreThresholdPercentage,
    int cfgScoreSimpleMovingAveragePeriod,
    int cfgUnsolicitedRunThreshold,
    char onlyTriggerOnPositiveDelta,
    const char* newScoreStr,
    const char* runSolutionScore,
    long * unsolicitedRunCounterPtr,
    char* normAppsSendCurrentCoresPtr,
    char reason[PVTM_REASON_SIZE],
    char backoffIfTooHigh,
    const char* appInfoJson,
    const char* appLinksJson
);
```

Here is a description of the parameters:

- `cfgMinScoreApplyPinningThreshold` – this allows PVTM Agent to suppress run solution requests to be sent unless the score is above a certain level. This is useful to suppress thread pinning automatically if the system is in an idle state.

- `cfgDeltaScoreThresholdPercentage` - The percentage (as an integer, so 50% would be 50) that the score has to differ from either the moving average or the last score to trigger a re-run.
- `cfgScoreSimpleMovingAveragePeriod` if this is greater than 0, it will make the callback look at a moving average rather than just the last score to determine when to trigger a re-run.
- `cfgUnsolicitedRunThreshold` - a threshold that if greater than 0 will cause a run request to be true regardless of the moving average every `cfgUnsolicitedRunThreshold` calls to this function.
- `newScoreStr` - a new score that reflects the current state of the system. This is a mutually exclusive value to the `runSolutionScore`. If `runSolutionScore` is not null, it overrides this value. This will also cause the moving average to be changed.
- `runSolutionScore` - if this is null, then the `newScoreStr` will be used; otherwise, this will make this function passively compare the score without modifying the moving average or last value.
- `unsolicitedRunCounterPtr` - an optional pointer to hold the current counter externally. If this is NULL, a local static variable will be used. Note that if not null, this should retain the counter across calls to this function.
- `normAppsSendCurrentCoresPtr` - this cannot be null! This is used to control whether or not we need to send the current cores where the threads are running as part of the normalized request.
- `reason` - this allows the user to get a textual description of why the simulation request was sent. This information is sent with the request, and is used for diagnostic purposes in the score time series.
- `backoffIfTooHigh` - Boolean variable that determines whether or not we should back out if there is a spike in the score greater than $\text{pvtm_agent_score_moving_average_period} * \text{pvtm_agent_delta_score_threshold_percentage} / 100$
- `appInfoJson` - JSON-style object with the latest SET_APPS data sent to the PVTM Thread Manager.
- `appLinksJson` - JSON-style object with the latest SET_APPLINKS data sent to the PVTM Thread Manager.

A sample makefile and 'hello world' implementations can be found under the following path:
 <installation directory root>/pontus-vision/4.7.0/linux/user-override

Command and Control

The command and control activities enable PVTM Agent to be remote-controlled for actions such as making it passive, active, getting the status, and even setting the configuration file contents. The command and control activities use two message types; the following sequence diagram shows the flow of these messages:

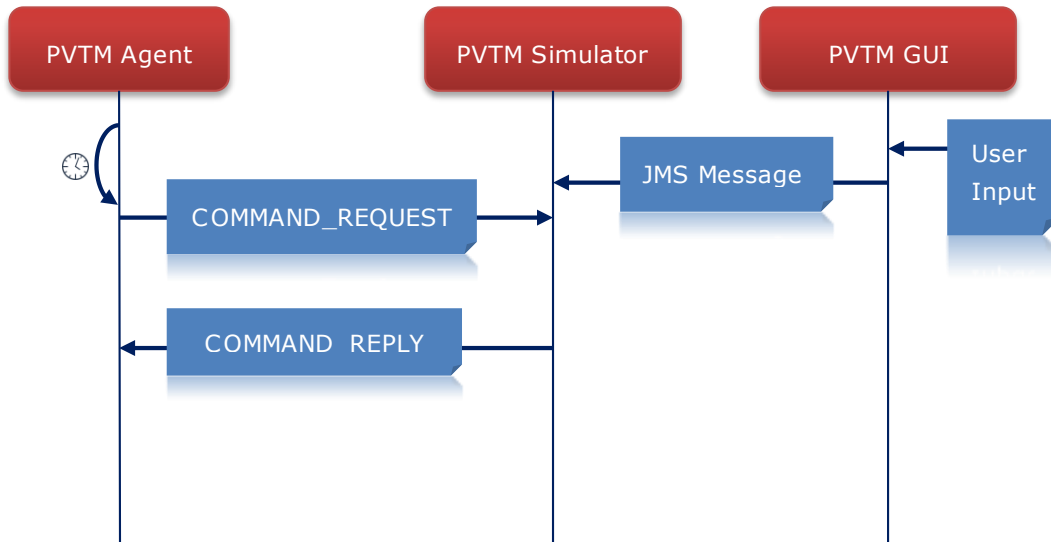


Figure 4 Command and Control Sequence Diagram

Note that if there are not any outstanding User Input requests, PVTM Agent ignores the COMMAND_REQUEST messages. The following sections go over the COMMAND_REQUEST and COMMAND_REPLY message formats:

COMMAND_REQUEST

PVTM Agent periodically sends the COMMAND_REQUEST message to PVTM Simulator to see whether any commands are outstanding.

The option `pvtm_agent_timer_poll_command_req_interval_ms` controls the frequency of these messages. The body of the request message simply has the word "GET" as a placeholder, and PVTM Simulator currently ignores it. The reply message can have the following types of command and control requests:

COMMAND_SET_PASSIVE

This puts PVTM Agent in passive mode, and causes it to send the status to PVTM Simulator via a COMMAND_REPLY message.

COMMAND_SET_ACTIVE

This command puts PVTM Agent in active mode, and causes the status to be sent back via a COMMAND_REPLY message.

COMMAND_RESET

This command puts PVTM Agent in passive mode, and resets all the thread pinning strategies, including for the junk cores. This causes PVTM Agent to send back a Passive Status via a COMMAND_REPLY message.

COMMAND_GET_CONFIG

This command reads the pvtm.cfg file contents, and sends it back to PVTM Simulator in a COMMAND_REPLY message.

COMMAND_GET_ACTIVE_STATUS

This command causes PVTM Agent to send the status back to PVTM Simulator in a COMMAND_REPLY message.

COMMAND_SET_CONFIG:SIZE=<size>:<data>

This command causes PVTM Agent to replace the contents of its pvtm.cfg file with the data sent. First, PVTM Agent writes a temporary configuration file in the same path as the pvtm.cfg file, but with a .tmp extension where it stores the data in transit. Once PVTM Agent receives the full data in the .tmp file, it backs up the original file into a new pvtm.cfg file with a <timestamp> suffix. Lastly, PVTM Agent copies the new data into a new pvtm.cfg file.

COMMAND_FORCE_RUN_SOLUTION_CACHED:SIZE=<size>:<data>

This command causes PVTM Agent to process the attached data and apply it as if it were a RUN_SOLUTION_CACHED response message.

COMMAND_TRAIN_FUZZY_CACHE:SIZE=<size>:<data>

This command causes PVTM Agent to process the attached data and apply it to build a fuzzy cache using an artificial neural network. The data has a series of inputs and outputs that enable PVTM Agent to train a fuzzy cache that helps speed up simulation results. The data can span several messages, and consists of a JSON-like object similar to the following:

```
{
  numThreads:3,
  numCores:2,
  params:
  {
    numEpochs:100000,
    numThreads: 1,
    algorithm: 2,
    learningRate: 0.35
    desiredError:0.01,
    bitFailLimit:0.09,
    clearPrevious:true,
  }
}
```

```

    cascadeTrainingMode: true,
    trainingArea:"./pvtm-agent-fuzzy-cache"
  }
  data:
  [
    {
      'i': [0,0,0,0, 0,0,0,0, 0,0,0,0],
      'o': [0,0,1]
    },
    {
      'i': [10,0,0,0, 10,0,0,0, 10,0,0,0],
      'o': [0,0,1]
    },
    {
      'i': [10,0,0,0, 10,0,0,0, 20,0,0,0],
      'o': [1,1,0]
    },
    {
      'i': [30,0,0,0, 10,0,0,0, 10,0,0,0],
      'o': [0,1,1]
    },
    {
      'i': [0,1,0,0, 0,1,0,0, 0,1,0,0],
      'o': [0,1,1]
    },
    {
      'i': [0,0, 10,1,0,0, 10,1,0,0, 10,1,0,0],
      'o': [0,1,1]
    },
    {
      'i': [10,1,0,0, 10,1,0,0, 20,1,0,0],
      'o': [0,0,1]
    },
    {
      'i': [30,1,0,0, 10,1,0,0, 10,0,0,0],
      'o': [1,0,0]
    }
  ]
}

```

Where:

- numThreads is the maximum number of threads in this batch of data
- numCores is the total number of cores in this batch of data (useful to ensure that the 'o' field in the output data is within range)
- parms is an optional object that has the following parameters that affect the training of the neural network:
 - numEpochs:100000 – this controls the maximum number of training cycles that PVTM Agent will use to start the neural network
 - numThreads: 1, - this controls the number of threads that PVTM Agent starts to train the neural network. The only supported value for this is currently 1

- algorithm: "RPROP" – this string parameter can be one of the following values:
 - 0 - INCREMENTAL, - Standard backpropagation algorithm, where the weights are updated after each training pattern. This means that the weights are updated many times during a single epoch. For this reason some problems will train very fast with this algorithm, while other more advanced problems will not train very well.
 - 1 - BATCH - this is a standard backpropagation algorithm, where the weights of the neural network are updated after calculating the mean square error for the whole training set. This means that the weights are only updated once during an epoch. For this reason some problems will train slower with this algorithm; however, since the mean square error is calculated more correctly than in incremental training, some problems will reach better solutions with this algorithm.
 - 2 - RPROP (The default) - this uses the rprop algorithm that automatically adapts to the problem and does not require other parameters such as the learning rate.
 - 3 - QUICKPROP - this is a more advanced batch training algorithm which achieves good results for many problems. The quickprop training algorithm uses the learningRate parameter along with other more advanced parameters, but it is only recommended to change these advanced parameters, for users with insight in how the quickprop training algorithm works. The quickprop training algorithm is described by [Fahlman, 1988]. ,
 - 4 - SARPROP – this is the SARPROP algorithm: a simulated annealing enhancement to resilient back propagation. See <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.8197&rep=rep1&type=pdf>)
- learningRate: 0.35 – the learning rate of the neural network; this value is ignored for the RPROP (default) algorithm.
- desiredError:0.01 – the desired error rate for the neural network.
- bitFailLimit:0.09 – the tolerance of the results (core number +/- bitFailLimit) that PVTM Agent will use whilst training the neural network.
- clearPrevious: true – determines whether or not the previous fuzzy cache should be cleared.
- cascadeTrainingMode: true – determines whether the fuzzy cache will be trained in cascade mode; cascade mode allows the fuzzy cache to dynamically grow hidden neural networks layers, whereas normal mode requires the neural network to be static. If this option is set to false, the fuzzy cache is created without any hidden layers, which reduces the size, but can also reduce the efficiency.
- trainingArea: :"/pvtm-agent-fuzzy-cache" – the location of the neural network on the PVTM Agent's local disk.
- data is an array of objects that have two arrays:
 - 'i' – input data; this integer array uses sets of 4 integers for each thread:
 - CPU % util,
 - the last core where the thread ran,
 - the number of connections to other threads,
 - the sum of all the weights across all the connections.
 - 'o' – output data; this integer array as the core number for each thread.

Note that to reduce the burden on the PVTM GUI due to the potentially large amounts of data, the PVTM Simulator is the one that queries the data locally, and sends the message above to PVTM

Agent once the query is processed. The message between PVTM GUI and PVTM Simulator only has the query parameters; it is a JSON-like structure similar to the following:

```
{
  leftTime: 1455098279772,
  rightTime: 1456098279772,
  maxEntries: 1000
  numEpochs:100000,
  numThreads: 1,
  algorithm: 2,
  learningRate: 0.35
  desiredError:0.01,
  bitFailLimit:0.09,
  clearPrevious:true,
  cascadeTrainingMode: true,
  trainingArea:"./pvtm-agent-fuzzy-cache"
}
```

Where:

- leftTime is the epoch value in milliseconds with the left-most boundry in the time series (the earlier timestamp) to be queried (inclusive).
- rightTime is the epoch value in milliseconds with the right-most boundry in the time series (the later timestamp) to be queried (inclusive).
- maxEntries caps the maximum number of data entries (see above) sent to train the Artificial Neural Network in the PVTM Agent.
- <all the other fields are the same as above>.

COMMAND_TRAIN_FUZZY_CACHE_STOP

This stops any fuzzy cache training in progress on the given PVTM Agent, and causes it to send the status to PVTM Simulator via a COMMAND_REPLY message.

COMMAND_QUERY_FUZZY_CACHE:SIZE=<size>:<data>

This command causes PVTM Agent to process the attached data and apply it to query a fuzzy cache using an artificial neural network. The data has a series of inputs for a single epoch point in time. It enables the PVTM GUI to verify the results that the given neural network will respond are valid, and indeed improve the performance. After the query is completed, PVTM Agent send the status to PVTM Simulator via a COMMAND_REPLY message.

The data in the COMMAND_QUERY_FUZZY_CACHE request message can span several messages, and consists of a JSON-like object similar to the following:

```
{
  trainingArea:"./pvtm-agent-fuzzy-cache",
  numThreads: 3,
  epoch: 1455066061526,
  i:[0,0,0,0, 0,0,0,0, 0,0,0,0],
  agentName: 'leo2-pc'
}
```

Where:

- trainingArea: :"/pvtm-agent-fuzzy-cache" – the optional location of the neural network on the PVTM Agent's local disk.
- numThreads: 3 – this has the number of threads included in the 'i' parameter below
- epoch: 1455066061526 – this has the timestamp in milliseconds since 1970 for this particular request.
- i:[0,0,0,0, 0,0,0,0, 0,0,0,0] - 'i' – input data; this integer array uses sets of 4 integers for each thread:
 - CPU % util,
 - the last core where the thread ran,
 - the number of connections to other threads,
 - the sum of all the weights across all the connections.
- agentName: 'leo2-pc' – an optional parameter that has the name of the agent that the data belongs to. This can be used if a 'foster' PVTM Agent is used to train the data on behalf of other agents.

COMMAND_QUERY_FUZZY_CACHE_STOP

This stops any fuzzy cache queries in progress on the given PVTM Agent, and causes it to send the status to PVTM Simulator via a COMMAND_REPLY message.

COMMAND_REPLY

PVTM Agent only sends a COMMAND_REPLY message to PVTM Simulator in reply to a COMMAND_REQUEST message. As such, the body of the COMMAND_REPLY messages always start with the name of the original command followed by either the token ":DATA:<data>", by the token ":ERROR:<error str>", or by the token "OK".

Here are a few examples:

- "COMMAND_SET_PASSIVE:OK" confirmation that .the set passive command was successfully executed; there are no error messages associated with this command.
- "COMMAND_RESET:OK" confirmation that .the reset command was successfully executed; there are no error messages associated with this command.
- "COMMAND_SET_ACTIVE:OK" confirmation that .the set active command was successfully executed; there are no error messages associated with this command.
- "COMMAND_GET_ACTIVE_STATUS:DATA:true" the current status is Active.
- "COMMAND_GET_ACTIVE_STATUS:DATA:false" the current status is Passive.
- "COMMAND_SET_CONFIG:ERROR:<desc>" Any errors found whilst creating the new configuration file or backing up the old one
- "COMMAND_SET_CONFIG:OK" confirmation that the command was successfully executed and that the new pvtm.cfg file is created, and the old one backed up.
- "COMMAND_GET_CONFIG:ERROR:<desc>" Any errors found whilst reading the existing pvtm.cfg file.
- "COMMAND_GET_CONFIG:DATA:<data>" The contents of the current pvtm.cfg file.
- "COMMAND_FORCE_RUN_SOLUTION_CACHED:OK" The new RUN_SOLUTION_CACHED_REPLY was successfully applied.
- "COMMAND_FORCE_RUN_SOLUTION_CACHED:ERROR:<msg>"
The new RUN_SOLUTION_CACHED_REPLY was NOT applied.

- "COMMAND_TRAIN_FUZZY_CACHE:OK"
The COMMAND_TRAIN_FUZZY_CACHE:SIZE=<size>:<data> message was fully received, and processing started.
 - "COMMAND_TRAIN_FUZZY_CACHE:ERROR:<desc>" Any errors found whilst processing the COMMAND_TRAIN_FUZZY_CACHE:SIZE=<size>:<data> message
 - "COMMAND_TRAIN_FUZZY_CACHE:UPDATE:<data>" Intermediate results of the fuzzy cache training, (e.g. "45%" – a short string with the percentage completed).
 - "COMMAND_TRAIN_FUZZY_CACHE:DATA:<data>" The results of the fuzzy cache training, including error rates, and whether the network was successfully trained.
 - "COMMAND_TRAIN_FUZZY_CACHE_STOP:OK" The fuzzy cache training was successfully stopped.
 - "COMMAND_QUERY_FUZZY_CACHE:DATA:<data>" The results of the fuzzy cache query, including the new thread pinning strategy positions for the initial query item. The data is in a JSON-like structure similar to this:


```
{
  numThreads: 3,
  epoch: 1455066061526,
  o: [2,1,20],
  agentName: 'leo2-pc',
  errorPcnt:0.0
}
```
- Where:
- numThreads is the number of threads in the reply
 - epoch is the timestamp in milliseconds of the original request
 - o is a JSON Array with integers representing the recommended cores for each of the input threads
 - agentName is an optional string that matches the original agent name in the request
 - errorPcnt is an optional percentage (from 0.0 to 1.0) indicating how much error the network currently has. This is not currently used, and is specified here for future implementations.
- "COMMAND_QUERY_FUZZY_CACHE:ERROR:<desc>" Any errors found whilst processing the queried data.
 - "COMMAND_QUERY_FUZZY_CACHE_STOP:OK" The fuzzy cache query was successfully stopped.

Command line utilities to send Control Messages

As of version 4.7.0.26, a basic command line utility was created to mimic the behaviour of the PVTM GUI. The PVTMAgentRemoteControl is able to send the COMMAND_REQUEST messages, above, and wait for the COMMAND_REPLY responses. Users are now able to externally perform batches of tests with different configurations without the GUI, or enable, disable, PVTM Agent, or even run ad-hoc simulations.

PVTMAgentRemoteControl

This utility enables users to dynamically send and receive commands to a specific PVTM Agent from the command line

Usage:

```
java -cp vision-server.jar com.pontusnetworks.utils.PVTMAgentRemoteControl
<agent name> <command> [<command data file>]
```

Where:

<agent name> is the name of the agent as seen in the GUI
<command> is the command to be sent e.g. (COMMAND_SET_PASSIVE,
COMMAND_RESET, COMMAND_SET_ACTIVE, COMMAND_SET_CONFIG,
COMMAND_GET_ACTIVE_STATUS, COMMAND_GET_CONFIG,
COMMAND_FORCE_RUN_SOLUTION_CACHED, COMMAND_TRAIN_FUZZY_CACHE,
COMMAND_TRAIN_FUZZY_CACHE_STOP, COMMAND_QUERY_FUZZY_CACHE,
COMMAND_QUERY_FUZZY_CACHE_STOP])
[<command data file>] is the optional file with the text to send as a part
of the command (no need to include the SIZE=<size> portion; the PVTM
Simulator does this already).

Note: to set the JMS broker name, or the topics to/from the PVTM Simulator, use the following JVM -D options before the class name:

```
java -Dpontus.agent.broker.name="tcp://localhost:61616" \  
-Dpvtm.threadmgr.to.gui.topic="pvtm.threadmgr.to.gui.topic" \  
-Dpvtm.threadmgr.from.gui.topic="pvtm.threadmgr.from.gui.topic" \  
-cp vision-server.jar com.pontusnetworks.utils.PVTMAgentRemoteControl \  
<agent name> <command> [<command data file>]
```

Also, note that it will usually take several seconds for a command to be successfully completed. This is primarily due to the current architecture, which was design to grow to hundreds of thousands of PVTM Agents at the cost of additional latency. PVTM Agent polls the PVTM Simulator periodically for command requests, and the requests made by this tool first go to the PVTM Simulator, and then to the agent once it requests a new command. This double cycle causes delays. PVTMAgentRemoteControl will block until a reply message arrives.

Here are some useful examples:

The following will set the configuration of agent leo2-pc, with the contents of the file windows/pvtm.cfg.

```
$ java -cp vision-server.jar com.pontusnetworks.utils.PVTMAgentRemoteControl
leo2-pc COMMAND_SET_CONFIG windows/pvtm.cfg
//# Received Reply message Agent (leo2-pc): command: COMMAND_SET_CONFIG;
status: OK
```

The following will get the current configuration of agent leo2-pc:

```
$ java -cp vision-server.jar com.pontusnetworks.utils.PVTMAgentRemoteControl
leo2-pc COMMAND_GET_CONFIG
```

The following will make PVTM agent leo2-pc passive:

```
$ java -cp vision-server.jar com.pontusnetworks.utils.PVTMAgentRemoteControl
leo2-pc COMMAND_SET_PASSIVE
```

```
//# Received Reply message Agent (leo2-pc): command: COMMAND_SET_PASSIVE;
status: OK
```

The following will make PVTM agent leo2-pc active:

```
$ java -cp vision-server.jar com.pontusnetworks.utils.PVTMAgentRemoteControl
leo2-pc COMMAND_SET_ACTIVE
//# Received Reply message Agent (leo2-pc): command: COMMAND_SET_ACTIVE;
status: OK
```

Terms and Concepts Definitions

To understand how PVTM Agent operates, it is important to recap the following terms and concepts:

Allowed Cores – the list of cores on which an application or process is allowed to run

App link – an inter-thread connection; the app links can be manually added, or automatically profiled by PVTM Agent.

Assume All Threads Talk – When this app property is set to true, the model will artificially add links between all the threads within an application. Users should only use this in environments where the inter-thread communication patterns are not automatically discovered.

Chrt – This Linux construct (**change real-time**) increases thread priority. The same setting also works for windows. PVTM increases the priority of any matching thread if running as root or administrator.

Context Switch Cost - the multiplier that the simulator will use to penalize the score for context switches. The formula used by the simulator is the following:

$$Pct = \frac{\ln(numThreads) * CPUUtil * ContextSwitchCost}{CpuScalingFactor}$$

Where:

numThreads is the number of threads currently scheduled in the core,

CPUUtil is the total CPU percentage of the threads currently scheduled in the core, or 0.01 if the total utilization is zero)

ContextSwitchCost is the value set in the SET_COMPUTER message

CPUScalingFactor is either 1, or the value set in the RUN_SOLUTION_CACHED request message

CPU Percentage Auto Passive – an automatic kill-switch to disable the thread manager pinning strategies after either the total CPU utilization for the whole machine, or a single core's CPU utilization increases beyond a certain level.

Ignore After Pinning – This construct makes PVTM Agent avoid run simulation requests for a grace period after it applies a new thread execution layout.

Inactive IRQs – On Linux, PVTM Agent can ignore IRQs that are not active.

Inflate CPU Threshold – This is a feature where PVTM Agent artificially inflates the total CPU utilization. PVTM Agent triggers this action if more than two threads are running in the same core, and their CPU utilization is greater than the threshold. This forces PVTM Simulator to try and separate out the threads into separate cores.

IRQ – Interrupt ReQuest – this is a Linux concept, where a device driver executes code to handle data coming in or out of a device. On Linux, users can achieve big performance gains by keeping the cores where the IRQs are executing as close to both the physical device location, and the application threads utilizing that device. As an example, a network card's IRQ should be pinned to run in the same CPU as the Network card, and the I/O application threads should be pinned to run as close to the device as possible.

IRQ Files – IRQs related to I/O for block devices (e.g. disk operations). PVTM Agent enables users to add rules that describe which files in the file system belong to which interrupt devices.

JSON – JavaScript Object Notation – this is the rough format used for the contents of some of the more complex messages coming to/from the simulator.

Junk Cores – Junk cores are cores dedicated to non-performance-sensitive applications. Background processes, including some Kernel threads run on junk cores to avoid interference with performance sensitive applications. PVTM Agent can move all non-matching processes, including itself to a list of Junk Cores. It is usually recommended that more than one junk core is used in case an application running in the junk cores uses 100% CPU utilization and renders the server inaccessible.

Lock Stat Countdown – When PVTM Agent profiles locks between threads, it has to be as unobtrusive as possible. To minimize the impact on the apps, PVTM Agents will just profile the first 'Lock Stat Countdown' calls every polling period.

Moving Average Period – the number of previous scores to be used in a moving average of scores. PVTM Agent keeps a rolling simple moving average of the last 'Moving Average Period' scores, and uses this number to determine whether a new thread pinning strategy is required.

Normalized App – PVTM applies a hash to RUN_SIMULATION_CACHED requests in an attempt to match previously simulated results. In order to be able to use this feature across servers, and for processes that have been recently created, the data needs to be stripped of process IDs, or any application-specific semantics. A Normalized app is an application definition that has had PIDs and TIDs replaced with an index number, so instead of having thread id 12345 and process ID 33234, the normalized app would have a thread id of 0, and a process ID of 0. For a more complete example, please compare the example SET_APPS message with the first part of the example RUN_SOLUTION_CACHED request message.

NUMA – Non-uniform memory access; a server architecture adopted by most major manufacturers where each CPU Socket in a server has its own local memory, but where all the memory is accessible via remote CPU Sockets; in NUMA architectures, accessing the local memory bank is significantly faster than accessing a remote memory bank.

Passive Pinning Mode – PVTM Agent is configured to start in passive pinning mode by default. When this is set, PVTM Agent will never act upon any automatic thread pinning strategies sent to it, and rather it will just collect data and send it back to the simulator.

Premium Cores – PVTM Agent considers premium cores any cores that are not Junk Cores. The simulator runs the applications that are performance-sensitive on Premium Cores.

Premium Junk Cores – This concept allows users to choose a sub-set of cores to run non-matching applications within the junk cores. This must always be a sub set of the Junk Cores. This is particularly useful if hyperthreading is turned on in a server, and a user wants to logically disable them. This can be done by setting the list of junk cores to all the hyper threads, plus a couple of real junk cores, and then setting the premium junk cores to just the real junk cores. This would prevent the simulator from using any hyper threads or the normal junk cores, and it would force PVTM Agent not to run any apps in the hyperthreads.

Proto Weights un_i4_i6_ld_pf_lk_sm – There are two proto weights arrays used to set the min and max values of inter-thread communication connection weights. These two arrays allow users to bias the weights of inter-thread connections. They contain the min and max values for the normalized weights of the following protocols:

- un - unknown protocol
- i4 - ipv4 sockets
- i6 - ipv6 sockets
- ld - local domain sockets
- pf - pipes and files
- lk - locks
- sm - shared memory

These arrays will spread the actual values for the protocols (the value in parenthesis of how many unlocks for locks, or bytes were sent on sockets between these artificial min and max values. The default values below will make all the unknown protocols with a weight of 1, all the ipv4,v6, and local domain sockets spread between 20 and 30, all the pipes and files between 1 and 5, all the locks at 1, and all the shared memory areas at 10. These will effectively bias the model towards favouring socket traffic over locks, pipes and files, and shared memory

Regex – short for Regular Expression – this is a very powerful language used extensively in PVTM Agent to filter out or match strings.

Run Solution Command – This is a string that PVTM Agent sends to the simulator whenever a RUN_SOLUTION_CACHED request is required. This option can control how the simulation will be run, including the amount of simulation time, as well as some of the algorithms used to place the threads into the cores in the model.

Same Core Cost – When PVTM Agent creates a SET_COMPUTER message, it builds a latency matrix with the distance between cores. The matrix works well for the distance between different cores; however, the distance between a core and itself is not computed, and is rather filled in by the same cores cost. If this value is too low, the model will tend to bias strong connections between threads to occur in the same core. This can impact the application performance by generating too many context switches. To avoid this, this value should be set to a high value. The only exception is virtual machines. When using VMs, the cost of context switches is not always as high as on a physical server. This is usually the case because virtual cores in a VM are really a software thread in the hypervisor. As such, the context switches in the VM do not necessarily cause a real full context switch, making the VM threads act as green threads. This, of course depends on the hypervisor and the application itself. It is recommended that thorough empirical tests are done before setting the same core cost to too low a value.

Thread pinning – Thread pinning is the action of telling the scheduler in which core(s) one or more threads are allowed to run.

Timer Poll Config File Interval – PVTM Agent can be configured to poll a configuration file periodically to look for any dynamic options. This is a mechanism to change rules of artificial inter-thread connections without stopping and starting PVTM Agent.

Timer Proc Interval – PVTM Agent uses several timers to poll for different types of information. The Timer Proc Interval is the interval, in ms, that PVTM Agent polls the operating system for new applications that match the regex filter.

Timer Thread Interval – Once PVTM Agent finds a list of processes that match the regex filter, it will introspect the threads for their CPU utilization. The Timer Thread Interval controls how often this happens. Making this value too long may makes the thread manager lose too many thread patterns; similarly, making this value too short may impact the performance of PVTM Agent, and make it too reactive to small changes in application behaviour.

Timer Command Request Interval –PVTM Agent periodically checks whether PVTM Simulator has any commands available to be sent to it. This timer controls how often the polling occurs. The commands can be putting the agent into an active or passive mode, creating or retrieving configuration files, or even forcing a RUN_SOLUTION_CACHED response message.

Unsolicited Run Threshold – PVTM Agent can be configured to send a RUN_SOLUTION_CACHED request message optionally without being triggered by any changes in scores. The unsolicited run threshold controls how often these unsolicited requests occur.

Web Sockets Payload – PVTM Agent uses web sockets to communicate to the simulator. The payload size determines how much data is allowed to be sent between them. In Applications that have very large lists of AppLinks, or very high number of threads, the size of the web sockets payload may need to be increased.

Zapping cores – PVTM Agent can, (currently only on Linux) clear out, or zap a number of cores to try and move non-essential threads from any given set of cores. Please be warned that this is a ****VERY INTRUSIVE**** option that should only be applied outside of production hours, and with a minimal number of processes running. The key benefit is that this enables low-latency applications to run with as few interruptions as possible.

Command Line

PVTM Agent is located in an operating-system specific folder inside the installation directory (e.g. /opt/pontus-vision/4.7.0/linux, or /opt/pontus-vision/4.7.0/windows). All the configuration files and libraries required are located within this directory; as such, the whole directory can be copied to servers that need to be optimised using any mechanism to copy the files.

Here are the command line arguments for PVTM Agent:

```
pvtm-agent  
[-r or --regex=<regular expression to match apps>]  
[-u <host> or --url=<url to connect (e.g.  
ws://localhost:8444/threadmgr/myhost)>]
```

```

[-c or --config=<conf>]
[-l or --load=[id,junkCoresInSquareBrackets,contextSwitchCost,
sameCoreCost,output.txt]
[-i <pid> or --inject=<pid> - inject the LD_PRELOAD behaviour on <pid>]
[-d or --dump (dump the shared memory area, and the cpu info and exit)]
[-t reset or --taskset='reset' - to unpin all processes, or
'premium:<pid>' to move pid and all its threads to the premium (non-junk)
cores,or
' [{"pid":9672,"threadIds":[9672,9684,9689],"coreIds":[0,4,4],"numaZoneIds":[
[0],[0],[0]],"score":"2217"} ] '

```

The following table shows the options in more detail:

Option Name	Default Value	Description
--regex	"" (empty string)	Controls the regular expression used by PVTM Agent to find applications that need to be optimized
--url or -u	"localhost"	Points PVTM Agent to a simulator process. This option can be set using two modes: the short notation uses a host name only, with a default port of 8444; the long version uses the full URI notation: ws://<sim host>:<port>/threadmgr/<agent host>
--config	"./pvtm.cfg"	This option selects the full path to a configuration file to be used to control various aspects of PVTM Agent. See the PVTM Agent Configuration section for more details
--load	"" (empty string)	Enables users to capture the server's model information for importing this server into the simulator. This option has a comma-separated list of items with the server id, the list of junk cores in square brackets, the same core cost, and the name of the output file to capture the data
--inject or -i	"" (empty string)	In windows only it enables PVTM Agent to profile a particular application; the value should be a valid PID
--ps_mode or -p	n/a	This dumps a list of all the inter-thread communication patterns in the system. Note that 64 bit applications can only be monitored by 64 bit versions of PVTM Agent, and similarly, 32-bit versions of PVTM Agent can only monitor 32-bit processes.
-t or -taskset	"" (empty string)	this option has two potential values; it can either be set to the word 'reset', or to a text representation of the RUN_SOLUTION_CACHED response message. When set to the word 'reset', PVTM Agent removes any thread affinity from all the applications that are running in the system.

In addition to the command line parameters above, the pvtm-agent executable on Linux also doubles as a completely different application when its name is changed to irq-helper. When a symbolic link with this name points to the pvtm-agent file, the irq-helper application can be run. IRQ helper is a utility that shows concise stats about interrupts that match a particular pattern. When in irq-helper mode, PVTM Agent uses the following command line arguments:

```

./irq-helper 'regex' <-c [max-cpu]|-s [num runs]>
Where:

```

```

'regex' is a regular expression that matches an IRQ
-c shows a pvtm config dump of the matching IRQs followed by an optional
max
  cpu util across all IRQ matches
-s (default) shows irq stats followed by an optional number of runs
(defaults
  to forever if blank)
(e.g. ./irq-helper 'eno|eth' )

```

The output of IRQ helper when in `-c` mode is a configuration compatible with the `pvtm_agent_irq_files` option. The output of IRQ helper when in `-s` mode has one or more the following lines with the following contents:

```

IRQ: 19 ← The number of the interrupt request handler
IO-APIC-fasteoi ← The type of interrupt
eno16777736: ← The device name
deltaIRQs = 1 ← The number of interrupts sent since the last line
lastCores = 0 ← The last core(s) where this IRQ ran
allowedCores = 0-1 ← The cores where this device is connected
cidrs=0.0.0.0/32,
      0.0.0.0/0,
      169.254.0.0/16,
      192.168.51.0/24 ← The routing table information associated with this
interrupt (if it is an IPv4 device)

```

Environment Variables

PVTM Agent uses a few environment variables to control various aspects of its operation. The following sections show the main environment variables and their use.

LD_PRELOAD

On Linux, PVTM Agent relies on applications to set the `LD_PRELOAD` environment variable so they can share key metrics used to find inter-thread communication patterns. Though PVTM Agent needs other apps to set this variable, this variable **MUST NOT BE SET** for PVTM Agent itself. A fail-safe mechanism prevents PVTM Agent from starting if the `LD_PRELOAD` environment variable contains the words "libpvtm-agent-preload"

PVTM_AGENT_CONFIG_FILE_NAME

The default name for the PVTM Agent's config file is `./pvtm.cfg`. This environment variable overrides this default value, allowing users to set it to any other name or path.

PVTM_AGENT_FILENAME_FORMAT

This environment variable controls the location of the shared memory area that PVTM Agent uses to communicate with applications using the `LD_PRELOAD` environment variable. Note that **IF THE**

DEFAULT VALUE IS NOT SUITABLE, THIS VARIABLE MUST BE SET ON BOTH PVTM AGENT AND THE APPLICATION BEING MONITORED.

The default location for the shared memory area is /tmp/pvtm_agent/shmem if this variable is not set.

PVTM_AGENT_FORCE_MUX_BUSY_POLL

This environment variable controls whether mutexes are marked for a busy poll, as well as the number of times the application will be spinning waiting for the mutex to unlock. This option only works on Linux applications using the LD_PRELOAD environment variable to load our shared library. The default value of either 0, or if the variable is unset means the user application will work normally. A value of -2 causes any applications calling pthread_mutex_lock to spin polling forever until the mutex is unlocked.

Any positive values are the number of times the application will poll the mutex waiting for it to be unlocked.

WARNING: this may significantly increase the CPU utilization of any threads calling the pthread_mutex_lock function, but should improve the application performance.

The following example will cause the thread calling mutex lock to spin forever:

```
export PVTM_AGENT_FORCE_MUTEX_BUSY_POLL=-2
```

PVTM_AGENT_FORCE_MUX_BUSY_POLL_FILTER

This environment variable has a | delimited list of strings that must be present in all the thread names that are supposed to use the busy_poll facility described above. If this option is not set, or set to an empty string, then all threads will do a busy spin when calling epoll_wait().

```
export PVTM_AGENT_FORCE_MUX_BUSY_POLL_FILTER="reader|writer"
```

PVTM_AGENT_FORCE_SO_BUSY_POLL

This environment variable controls whether sockets are marked for a busy poll, as well as the number of times the application will be spinning waiting for traffic to arrive in a socket. This option only works on Linux applications using the LD_PRELOAD environment variable to load our shared library. The default value of either 0, or if the variable is unset means the user application will work normally. A value of -1 causes any applications calling epoll_wait to spin polling for data to be available from the socket ignoring the timeout argument sent in the epoll_wait API. A value of -2 will make the timeout variable be honoured during the spin, causing the application to only spin for the timeout period.

Any positive values are the number of times the application will poll the epoll_wait function waiting for data to arrive.

WARNING: this will significantly increase the CPU utilization of any threads calling the epoll_wait function, but should improve the application performance.

The safest value when using this option is to set it to the adaptive mode (-2) :

```
export PVTM_AGENT_FORCE_SO_BUSY_POLL=-2
```

PVTM_AGENT_FORCE_SO_BUSY_POLL_FILTER

This environment variable has a | delimited list of strings that must be present in all the thread names that are supposed to use the busy_poll facility described above. If this option is not set, or set to an empty string, then all threads will do a busy spin when calling epoll_wait().

```
export PVTM_AGENT_FORCE_SO_BUSY_POLL_FILTER="reader|writer"
```

PVTM_AGENT_SHMEM

This environment variable enables users to select which directory will be used for the shared memory area files used by PVTM Agent to gather stats from user applications. The default value on Linux is /tmp/pvtm_agent/shmem/, whereas on Windows it is c:\Pontus\pontus-vision\4.7.0

PVTM_DISABLE_HUGE_PAGES

This environment variable disables the use of huge pages on Linux. It is useful if an error message complaining of no "TLS Huge Pages found" appears, in which case this should be set to 1 as follows:

```
export PVTM_DISABLE_HUGE_PAGES=1
```

Note: This is required when using Informatica's UMP stores and LD_PRELOAD, or the ONLOAD_PRELOAD settings.

PVTM_DISABLE_NUMACTL_LOCAL

This environment variable disables the use of local memory allocation on Linux. It is useful if a behaviour other than numactl --localalloc is desired. Setting this variable to 1 should disable this:

```
export PVTM_DISABLE_NUMACTL_LOCAL=1
```

PVTM_ENABLE_SELF_PINNING

This environment variable controls whether the LD_PRELOAD library on Linux will attempt to look for a previous pinning strategy result saved by PVTM Agent when the option pvtm_agent_store_last_core_on_command_force_run_solution_cached is set. If this option is empty, or not set, this feature is disabled. This currently only works on Linux.

PVTM_PID_FILE_DIR

On Linux only, PVTM Agent uses a PID file to prevent more than one process running on the same server. The default value for the directory where this file resides is to look up the environment variable PVTM_PID_FILE_DIR. If this variable is not set, the default is the HOME environment variable.

pvtm.cfg Configuration File

PVTM Agent is configured by using a pvtm.cfg file that usually resides in the same directory as PVTM Agent itself. The pvtm.cfg file can refer to the pvtm.cfg files by using the @include "file name" directives. The pvtm.cfg file will always use the latest instance of a variable, so if an @include directive is added to the top of the file, individual options can be overridden below that.

BNF Grammar

Comments and include directives are not part of the grammar, so they are not included here. Comments follow the 'C++' and 'Java' language formats. Single-line comments start with two slashes (//), and go to the end of the line. Multi-line comments start with a slash and a star (/*) and end with the first (*).

Below is the Backus-Naur Form (BNF) grammar for configuration files:

```
configuration = setting-list | empty
setting-list = setting | setting-list setting
setting = name (":" | "=") value (";" | "," | empty)
value = scalar-value | array | list | group
value-list = value | value-list "," value
scalar-value = Boolean | integer | integer64 | hex | hex64 | float
              | string
scalar-value-list = scalar-value | scalar-value-list "," scalar-value
array = "[" (scalar-value-list | empty) "]"
list = "(" (value-list | empty) ")"
group = "{" (setting-list | empty) "}"
empty =
```

Scalar values are defined below as regular expressions:

```
Boolean      ([Tt][Rr][Uu][Ee])|([Ff][Aa][Ll][Ss][Ee])
string       \(^[^"\\\]|\\.)*\`
name         [A-Za-z\]*[-A-Za-z0-9_\`]*
integer      [-+]?[0-9]+
integer64    [-+]?[0-9]+L(L)?
hex          0[Xx][0-9A-Fa-f]+
hex64        0[Xx][0-9A-Fa-f]+(L(L)?)?
float        ([-+]?([0-9]*)?\.[0-9]*([eE][-+]?[0-9]+)?)|([-+]?([0-9]+)(\.[0-9]*)?[eE][-+]?[0-9]+)
```

Configuration Values

The following sections describe the configuration values in the pvtm.cfg file in alphabetical order:

pvtm_agent_aggressive_run_solution_if_score_too_high

This Boolean option controls whether the agent re-requests a new simulation if the score from the previous simulation results was higher than the latest score in the system. **WARNING:** if users set this value to true, too much re-pinning, which can impact the application performance, can occur. The default value for this option is false.

pvtm_agent_app_config_override

This 'list of groups' option enables PVTM Agent to override application characteristics, such as the set of cores where the app is allowed to run, or any other attributes in the SET_APPS message. As listed in the BNF Grammar, the list uses parenthesis with one or more groups of settings. Each of the internal groups is delimited by curly braces {}, and is comma-delimited. The internal groups show different rules to override application characteristics, and have the following options:

- `regex` – this string option has a regular expression that is used to find application names for the current override rule. Any matching apps will have the other attributes overridden. Note that the matching starts at the first internal group, and continues downwards. The first match found stops the traversal.
- `overrideStr` – this string option has embedded JSON-like objects that are appended to the normal matching SET_APPS application descriptions sent to PVTM Simulator. It must be present, even if it is empty.
- `threadAllowedCores` - this is an optional value-list option of value-list options (a 2-dimensional array) of strings with regex patterns and strings with allowed cores for any threads that match that pattern (e.g. `threadAllowedCores=((("thread_2","[-1]"),("java","[0,2]"));`). The value of the strings with the cores must always have square brackets, and one or more integers. To specify that a thread should run on all cores, a special value of `"[-1]"` may be used.

The following example shows three rules to override application behaviour. The first rule overrides any applications that match the pattern `app2` to be allowed only to run on cores 1 and 3. The second rule overrides any apps that match the pattern `app1` to only be allowed to run on cores 1, 2 and 3 (**note that PVTM does a conservative match, quitting after the first match is found, so an app that matched both patterns would end up only in cores 1 and 3**). Lastly, the third rule overrides any applications that match the pattern `java`, allowing any threads that match the thread name `thread_2` to run on any cores (the `-1` means all cores), and any threads named `java` to only run on cores 0 or 2

```
pvtm_agent_app_config_override=  
(  
  {regex="app2"; overrideStr="'allowedCores':[1,3]";},  
  {regex="app1"; overrideStr="'allowedCores':[1,2,3]";},  
  {  
    regex="java";  
    threadAllowedCores=((("thread_2","[-1]"),("java","[0,2]"));  
    overrideStr = " ";  
  }  
)
```

);

pvtm_agent_app_link_file_delta_threshold

This integer option enables PVTM Agent to filter out inter-thread communications that use files or named pipes that are not sending enough data. If this option is set to a value of 0, any files or pipes that have more than zero bytes written or read are included in the SET_APPLINKS message. If it is set to 1000, then only files or pipes that had more than 1000 bytes since the last polling interval are added to the SET_APPLINKS message. If this option is set to -1, then all pipe and file connections (whether or not data has been sent) are added to the SET_APPLINKS message.

The default value for this option is `pvtm_agent_app_link_file_delta_threshold = 0;`

pvtm_agent_app_link_file_delta_threshold_bypass_pre_filter

This Boolean option enables PVTM Agent to bypass the checks for the files and pipes threshold set in `pvtm_agent_app_link_file_delta_threshold`. If set to true, then no threshold checks are performed.

The default value for this option is `pvtm_agent_app_link_file_delta_threshold_bypass_pre_filter =false`

pvtm_agent_app_link_inclusive_regex_filter

This string option enables PVTM Agent to filter which app link strings are included in the SET_APPLINKS message sent to the PVTM Simulator. Only entries that match this option are included. The default value is to match all entries.

pvtm_agent_app_link_inet6_socket_delta_threshold

This integer option enables PVTM Agent to filter out inter-thread communications that use IPV6 sockets that are not sending enough data. If this option is set to a value of 0, any IPV6 socket that has more than zero bytes will be included in the SET_APPLINKS message. If it is set to 1000, then only IPV6 sockets that had more than 1000 bytes since the last polling interval will be added to the SET_APPLINKS message. If this option is set to -1, then all IPV6 socket connections (whether or not data has been sent) are added to the SET_APPLINKS message.

The default value for this option is `pvtm_agent_app_link_inet6_socket_delta_threshold = 0;`

pvtm_agent_app_link_inet6_socket_delta_threshold_bypass_pre_filter

This Boolean option enables PVTM Agent to bypass the checks for the inet6 threshold set in `pvtm_agent_app_link_inet6_socket_delta_threshold`. If set to true, then no threshold checks are performed.

The default value for this option is

`pvtm_agent_app_link_inet_socket_delta_threshold_bypass_pre_filter =false`

pvtm_agent_app_link_inet_socket_delta_threshold

This integer option enables PVTM Agent to filter out inter-thread communications that use IPV4 sockets that are not sending enough data. If this option is set to a value of 0, any IPV4 socket that has more than zero bytes will be included in the SET_APPLINKS message. If it is set to 1000,

then only IPV4 sockets that had more than 1000 bytes since the last polling interval will be added to the SET_APPLINKS message. If this option is set to -1, then all IPV4 socket connections (whether or not data has been sent) are added to the SET_APPLINKS message.

The default value for this option is `pvtm_agent_app_link_inet_socket_delta_threshold = 0;`

pvtm_agent_app_link_inet_socket_delta_threshold_bypass_pre_filter

This Boolean option enables PVTM Agent to bypass the checks for the inet threshold set in `pvtm_agent_app_link_inet_socket_delta_threshold`. If set to true, then no threshold checks are performed.

The default value for this option is

`pvtm_agent_app_link_inet_socket_delta_threshold_bypass_pre_filter = false`

pvtm_agent_app_link_lock_delta_threshold

This integer option enables PVTM Agent to filter out inter-thread communications that use futex locks that are not sending enough data. If this option is set to a value of 0, any futex locks that have been activated at least once since the last polling interval will be included in the SET_APPLINKS message. If it is set to 1000, then only futex locks that had more than 1000 locks and unlocks since the last polling interval will be added to the SET_APPLINKS message. If this option is set to -1, then futex locks (whether or locking/unlocking have been done) are added to the SET_APPLINKS message.

The default value for this option is `pvtm_agent_app_link_lock_delta_threshold = 0;`

pvtm_agent_app_link_lock_delta_threshold_bypass_pre_filter

This Boolean option enables PVTM Agent to bypass the checks for the locks threshold set in `pvtm_agent_app_link_lock_delta_threshold`. If set to true, then no threshold checks are performed.

The default value for this option is

`pvtm_agent_app_link_lock_delta_threshold_bypass_pre_filter = false`

pvtm_agent_app_link_override

This "list of groups" option enables PVTM Agent to create artificial inter-thread application links that will appear as 'unknown' protocols in the SET_APPLINKS message.

As listed in the BNF Grammar, the list uses parenthesis with one or more groups of settings. Each of the internal groups is delimited by curly braces {}, and is comma-delimited. The internal groups show different rules to add new inter-thread application links, and have the following options:

- (source/dest)AppRegex – this string option is a regular expression used to match one or more applications to act as the (source/dest) of the connection
- (source/dest)AppRegexIndex – this integer option is used to determine which of the regex matches is used; when set to -1, all the regex matches are used.
- (source/dest)ThreadIndex – this integer option is one of the two mechanisms used to determine which thread within each matching application is used as the (source/dest) of the connection. If this is not present, the second mechanism (regex on the thread name) is used to identify the threads.

- (source/dest)ThreadNameRegex – this string option is a regular expression used to match one or more thread names as the (source/dest) thread of the connection
- (source/dest)ThreadNameRegexIndex – this integer option is used to determine which of the regex matches is used; when set to -1, all the regex matches are used, resulting in multiple connections.

Here is a brief example of two rules that create a link between app2(t2), and app1(t1), and a link between the first and second threads in app1:

```
pvtm_agent_app_link_override =
(
{ sourceAppRegex="app2"; sourceAppRegexIndex=0; sourceThreadNameRegex="t2";
sourceThreadNameRegexIndex=0; destAppRegex="app1"; destAppRegexIndex=0;
destThreadNameRegex="t1"; destThreadNameRegexIndex=0; },

{ sourceAppRegex="app1"; sourceAppRegexIndex=0; sourceThreadIndex=0;
destAppRegex="app1"; destAppRegexIndex=0; destThreadIndex=1; }

);
```

pvtm_agent_app_link_pair_delta_pcmt_threshold

This integer option causes PVTM Agent to only report a link between threads if the difference between the send and receive stats is less than this amount. The default value is 1000, which means 1000%. If a few threads are reporting a lock to a mutex, that lock is only reported as being a link if the number of locks/unlocks is within 1000% of the highest lock number reported.

pvtm_agent_app_link_poll_with_thread_info

This Boolean option controls when PVTM Agent polls the application link information from the applications being monitored. By default, PVTM Agent polls the application link information at the same frequency as the pvtm_agent_timer_poll_proc_interval_ms timer. If this option is set to true, PVTM Agent polls the application link information at the same frequency as the pvtm_agent_timer_poll_thread_interval_ms timer. The default setting is coarser, which means that fewer changes in the inter-thread patterns will occur; however, that also introduces a larger lag between a change in thread patterns and the reaction to them by re-pinning the process. Setting this option to true will typically cause more volatility in the PVTM scores.

The default value for this option is pvtm_agent_app_link_poll_with_thread_info = false;

pvtm_agent_app_link_unix_socket_delta_threshold

This integer option enables PVTM Agent to filter out inter-thread communications that use Unix domain sockets that are not sending enough data. If this option is set to a value of 0, any Unix domain socket that has more than zero bytes will be included in the SET_APPLINKS message. If it is set to 1000, then only Unix domain sockets that had more than 1000 bytes since the last polling interval will be added to the SET_APPLINKS message. If this option is set to -1, then all Unix domain socket connections (whether or not data has been sent) are added to the SET_APPLINKS message.

The default value for this option is `pvtm_agent_app_link_unix_socket_delta_threshold = 0;`

pvtm_agent_app_link_unix_socket_delta_threshold_bypass_pre_filter

This Boolean option enables PVTM Agent to bypass the checks for the Unix domain sockets threshold set in `pvtm_agent_app_link_unix_socket_delta_threshold`. If set to true, then no threshold checks are performed.

The default value for this option is

`pvtm_agent_app_link_unix_socket_delta_threshold_bypass_pre_filter=false`

pvtm_agent_app_link_use_preload

This Boolean option controls whether or not PVTM Agent should poll a shared memory area used by PVTM to keep counters indicating how many times each thread has accessed a select subset system calls.

On Linux, the method currently used to gather this information is to set the LD_PRELOAD environment variable to the following value before starting each process that needs to be profiled:

```
export LD_PRELOAD=<INSTALL_PATH>/linux/'$LIB'/libpvtm-agent-preload.so
```

(replacing `<INSTALL_PATH>` with the installation path up to the version number) for example:

```
export LD_PRELOAD=/opt/pontus-vision/4.7.0/linux/'$LIB'/libpvtm-agent-preload.so
```

Note that the `'$LIB'` should be kept as it is with the single quotes, as the Linux loader will dynamically replace it with either `lib` or `lib64` depending on whether the application being run is 32 or 64 bits.

On Windows, PVTM agent will automatically gather this information from the applications that match the [pvtm_agent_regex_pattern](#) by setting up a high-performance debugging session. As such, the applications being monitored must run by a user that has either debugging privileges enabled, or full admin rights.

The default value for this option is true on Linux, and false on Windows.

pvtm_agent_app_link_use_preload_linux_inject

This Boolean option controls whether or not PVTM Agent on Linux will attempt to profile remote applications without using LD_PRELOAD. The default value is false. Users that want to try this new experimental behaviour must also set the `pvtm_agent_app_link_use_preload` option to true. Note that applications should NOT use both LD_PRELOAD and this method simultaneously.

pvtm_agent_app_thread_perf_counter_events_csv

This string option controls whether PVTM Agent starts capturing performance counters kept by the Linux Kernel (2.6.36+). If this option is set to a comma-separated-value (csv) string, it causes the PVTM Agent to add two extra fields in the SET_APPS message sent to PVTM Simulator (`pvtmPerfCounterVals`, and `pvtmPerfCounterEventsCsv`). The following example shows the

number of Level 1 data cache misses, Last Level Cache (LLC) -- typically level 3 -- data cache misses, and data Translate lookaside buffer (TLB) misses:

```
pvtm_agent_app_thread_perf_counter_events_csv="L1-dcache-load-misses,L1-dcache-store-misses,LLC-load-misses,LLC-store-misses,dTLB-load-misses,dTLB-store-misses"
```

This option is currently only available on Linux, and the values are only used in "The Data Analysis Preview Area".

The default value (if left unset) is to disable this feature. **WARNING:** setting this to an empty string will not disable this feature; to disable this, simply leave this option commented out.

pvtm_agent_assume_all_threads_talk

This Boolean option controls whether PVTM Agent tells PVTM Simulator to assume that all the threads within an application talk to each other. Note that all platforms now have mechanisms to determine the inter-thread communication patterns, so this option should be switched off unless it is impossible to employ the supported discovery mechanism for inter-thread communication.

For legacy reasons, the default is true on all platforms.

pvtm_agent_chrt_rr_matching_processes

This Boolean option controls whether or not PVTM Agent should change the scheduler priority mechanism of all matching threads to the value set by the option `pvtm_agent_set_chrt_rr_priority`. The name `chrt` is a linux construct (**change real-time**) that enables the priority of the threads to be increased in real-time. The same setting also works for windows. PVTM increases the priority of any matching thread if running as root or administrator.

The default value for this option is `pvtm_agent_chrt_rr_matching_processes=true`;

WARNING: you must run `pvtm-agent` as root for this to work. If you do not run the agent as root, this option should be switched off; otherwise, error messages will occur.

pvtm_agent_context_switch_cost

This integer option controls the cost of context switch send to the simulator. If this number is too low, the model may place too many threads on the same core. The context switch cost is the multiplier that the simulator uses to penalize the score for context switches. The formula used by the simulator is the following:

$$Pct = \frac{\ln(numThreads) * CPUUtil * ContextSwitchCost}{CpuScalingFactor}$$

Where:

- numThreads is the num of threads currently scheduled in the core,
- CPUUtil is the total CPU percentage of the threads currently scheduled in the core, or 0.01 if the total util is zero)
- ContextSwitchCost is the value set in the SET_COMPUTER message

CPUScalingFactor is either 1, or the value set in the RUN_SOLUTION_CACHED request message

The default value is `pvtm_agent_context_switch_cost=100;`

pvtm_agent_cpu_info_json

This string option has a JSON-like structure that forces PVTM Agent to use an artificial hardware configuration. The format of this value is the same as the RUN_SOLUTION_CACHED request message.

Here are some examples:

```
//pvtm_agent_cpu_info_json="{ 'type': 'pontusdraw2d.IBM_3750_m4',
'properties': { 'id': 'zeus', 'contextSwitchCost': '40', 'junkCores':
'[]', 'physicalCpuList' : '[0,0,0,0,0,0,0,0]', 'latencyMatrix':
'[[[14,20,19,20,10,20,19,20], [20,14,20,19,20,10,20,19],
[19,20,14,20,19,20,10,20], [20,19,20,14,20,19,20,10],
[10,20,19,20,14,20,19,20], [20,10,20,19,20,14,20,19],
[19,20,10,20,19,20,14,20], [20,19,20,10,20,19,20,14]]' } } "
// pvtm_agent_cpu_info_json="{ 'type': 'pontusdraw2d.IBM_3750_m4',
'properties': { 'id': 'zeus', 'contextSwitchCost': '40', 'junkCores':
'[]', 'physicalCpuList' : '[0,0,0,0,0,0,0,0]', 'latencyMatrix':
'[[[40,20,19,20,10,20,19,20], [20,40,20,19,20,10,20,19],
[19,20,40,20,19,20,10,20], [20,19,20,40,20,19,20,10],
[10,20,19,20,40,20,19,20], [20,10,20,19,20,40,20,19],
[19,20,10,20,19,20,40,20], [20,19,20,10,20,19,20,40]]' } } "
```

pvtm_agent_cpu_max_total_pcmt_auto_passive

This integer option controls whether or not the agent becomes passive after the total CPU percentage across all the threads that match the [pvtm_agent_regex_pattern](#) in the whole server goes beyond this value (post scaling).

The default value of `pvtm_agent_cpu_max_total_pcmt_auto_passive = -1` disables this option.

pvtm_agent_cpu_scaling_factor

This numeric option enables PVTM Agent to get sub-one % cpu utilizations. For performance reasons, the agent deals with integers when sending the data to the simulator. A scaling factor of 10 sets the minimum CPU utilization to 0.1% a scaling factor of 100, 0.01. The default of 1 sets the minimum CPU utilization to 1%.

WARNING: if using this option, and hand-entering IRQ information using the option `pvtm_agent_irq_files`, please remember to multiply the threadPcnt number by the new scaling factor!

The default value is `pvtm_agent_cpu_scaling_factor=1;`

pvtm_agent_cpu_single_core_pcmt_auto_passive

This integer option controls whether or not the agent becomes passive after the CPU percentage in any core exceeds this value (from 0 to 100) post scaling.

The default value of `pvtm_agent_cpu_single_core_pcmt_auto_passive = -1` disables this option.

pvtm_agent_cpu_state_linux

pvtm_agent_cpu_state_linux_percentage_threshold

This integer option controls the cpu state of the server. When set to zero, the CPUs will never sleep; when set to -1, the system defaults to the current behaviour, and this becomes a no-op. When set to -2, the system will attempt to look at the available turbo frequencies, and select an appropriate CPU state. If various turbo frequencies are available in the system, sometimes setting the CPU state to 0 will actually reduce the maximum possible turbo frequency. As an example, in a Sandy Bridge EP system with an 8-core E5 2680 CPU, the following turbo frequencies are available:

- 3100 MHz max turbo 8 active cores
- 3100 MHz max turbo 7 active cores
- 3200 MHz max turbo 6 active cores
- 3200 MHz max turbo 5 active cores
- 3200 MHz max turbo 4 active cores
- 3400 MHz max turbo 3 active cores
- 3500 MHz max turbo 2 active cores
- 3500 MHz max turbo 1 active cores

In the example above, if only one or two cores are active, setting the CPU state to 0 will cap the maximum frequency available to 3.1GHz, potentially leading to over 10% reduction in max frequency of 3.5GHz for 1 or two active cores. When the `pvtm_agent_cpu_state_linux` is set to -2, PVTM Agent will check the number of cores that are active above the `pvtm_agent_cpu_state_linux_percentage_threshold` utilization, and then only set the CPU state to zero if there are either zero busy cores, or too many busy cores. With zero busy cores, the apps are usually not polling for data, and thus can really benefit from having the extra boost that a CPU State of 0 will give when they are awakened by new data. If there are more than 7 busy cores in the example above, the max CPU speed is the same as setting all cores to a CPU state of 0 anyway. To avoid flip-flopping between CPU states, PVTM Agent keeps the moving average of the last `pvtm_agent_score_moving_average_period` CPU percentages for each of the cores.

As the name implies, these options only impact Linux servers. These are a great options for reducing latency, but they can increase the power consumption of inactive servers.

The default values are the following:

`pvtm_agent_cpu_state_linux = -2,`
`pvtm_agent_cpu_state_linux_percentage_threshold=`

$$\frac{\text{MaxTurboMHz}_{\text{core 1}} - \text{MaxTurboMHz}_{\text{core n}}}{\text{MaxTurboMHz}_{\text{core n}}} * 100$$

where:

`MaxTurboMHzcore 1` is the maximum turbo frequency for 1 active core,
`MaxTurboMHzcore n` is the maximum turbo frequency for all active cores

If setting the `pvtm_agent_cpu_state_linux_percentage_threshold` option by hand, use whole numbers from 0 to 100 (note that this value is not affected by the option `pvtm_agent_cpu_scaling_factor`). The `pvtm_agent_cpu_state_linux_percentage_threshold` option has no impact unless `pvtm_agent_cpu_state_linux` is set to -2.

`pvtm_agent_delta_cpu_util_threshold`

This integer option controls the threshold in CPU utilization that will trigger a re-score. PVTM Agent sends score requests whenever the CPU utilization of any of the threads changes by `pvtm_agent_delta_cpu_util_threshold` percentage, or whenever a new thread starts, or PVTM Agent can also be configured to force a score request to be sent periodically regardless of any changes in the environment. The `pvtm_agent_delta_cpu_util_threshold` option is always an integer greater than or equal to zero. Setting the `pvtm_agent_delta_cpu_util_threshold` option to 5 means scores will be sent whenever the difference in CPU utilization on a thread was more than 5%.

The default value is `pvtm_agent_delta_cpu_util_threshold=5`;

`pvtm_agent_delta_score_threshold_percentage`

This integer option controls the difference in the moving average to trigger a new thread pinning strategy; note that if you set this to -1, PVTM Agent runs in passive mode, only capturing data, but never applying it.

The default value is `pvtm_agent_delta_score_threshold_percentage=10`; to make PVTM Agent run in passive mode, it can be set to `pvtm_agent_delta_score_threshold_percentage=-1`;

`pvtm_agent_delta_score_threshold_positive_trigger_only`

This Boolean option configures PVTM Agent to only send re-pinning requests if the difference between the current score and the moving average is a positive value (e.g. if the score increased). If set to false, both positive and negative changes will trigger a re-pinning strategy.

The default value is `pvtm_agent_delta_score_threshold_positive_trigger_only=true`;

`pvtm_agent_delta_time_to_delete_old_files_ms`

This integer option controls how long (in milliseconds) PVTM Agent waits to delete a shared memory area that has not been accessed recently. A shared memory area is only deleted if it has not been updated for longer than this value.

The default value is `pvtm_agent_delta_time_to_delete_old_files_ms=30000`

pvtm_agent_dynamic_config_file_name

This string option sets the name of the dynamic config file polled by PVTM Agent. PVTM Agent can poll a dynamic configuration file periodically (every `pvtm_agent_timer_poll_cfg_file_interval_ms`) to look for any dynamic options. This is a mechanism to change rules of artificial inter-thread connections without stopping and starting PVTM Agent. This option sets the full path to the file name being polled. If unset, PVTM Agent will never poll the dynamic configuration file. The default value is for `pvtm_agent_dynamic_config_file_name` to be unset.

pvtm_agent_fake_zero_percentage_threads_last_core

This Boolean option instructs PVTM Agent to fake the last core where a 0% thread was running rather than use the value reported by the OS. Threads do not respond to thread pinning requests until they execute their next instruction, so 0% threads are often reported as running in a different core to the one they were pinned, because that's the last core where they ran. This can cause unnecessary re-pinning requests to be made.

The default value for this option is true;

pvtm_agent_force_run_solution_request_if_new_threads_start

This Boolean option instructs PVTM Agent to send an unsolicited `RUN_SOLUTION_CACHED` request whenever a new thread appears.

The default value for this option is the following:

```
pvtm_agent_force_run_solution_request_if_new_threads_start = false;
```

pvtm_agent_ignore_inactive_irqs

This Boolean option instructs PVTM Agent to ignore inactive IRQs. This option currently only has an impact on Linux. It forces PVTM's IRQ auto-discovery mechanism to report only active interrupts.

The default value for this option is `pvtm_agent_ignore_inactive_irqs=true`;

pvtm_agent_ignore_zero_percentage_threads

This Boolean option enables PVTM Agent to ignore any threads that only use zero % CPU. This may be useful in situations where there is a very large number of threads; however, the down-side is that since the zero percent threads will not belong to the model, they will not be pinned properly, and may interfere, when they do wake up, with the high-performance threads.

The default value is `pvtm_agent_ignore_zero_percentage_threads=false`;

pvtm_agent_inflate_cpu_util_threshold

This integer option causes PVTM Agent to inflate the CPU utilization of threads that are running on the same core and that are perhaps limited in their performance by each other. If the CPU utilization of more than one thread is greater than or equal to the

`pvtm_agent_inflate_cpu_util_threshold`, the first thread's CPU utilization is reported normally; however, the next thread's CPU utilization is inflated to add up to 101%. This immediately causes PVTM Agent to send a `STOP_SOLUTION` request, which forces a new thread simulation to be calculated. This option only inflates the CPU utilization when more than one thread running on a single core.

The default value for this option is `pvtm_agent_inflate_cpu_util_threshold= -1`, which disables this feature.

`pvtm_agent_invalid_solution_counter_threshold`

This integer option controls the number of invalid solutions that PVTM Agent receives in a row before it gives up requesting a new pinning strategy.

The default value for this option is `pvtm_agent_invalid_solution_counter_threshold=25`.

`pvtm_agent_irq_auto_filter_regex_pattern`

This string option allows PVTM Agent to automatically discover IRQs on Linux, and use them in the model. Any interrupts in the `/proc/interrupts` file that match this value are used. A good way to test values for this option this is to use the PVTM Agent in 'irq-helper' mode as described in the Command Line section. This option only has an impact Linux platforms.

The default value is an empty string, which disables this feature.

`pvtm_agent_irq_auto_force_allowed_cores`

This string option has a JSON-style array of integers that overrides the allowed cores on where the automatically discovered interrupts on Linux can run. This is useful to anchor the interrupts in a subset of cores of the total number of cores in the whole CPU where the device is located. If this option is not set, PVTM Agent will automatically allow the IRQs to run on any of the cores of the CPU where the device is located (if it can locate the device). If the device cannot be located, the interrupts are allowed to run in the whole server.

The default value is an empty string, which disables this feature.

`pvtm_agent_irq_auto_max_total_cpu_util`

This integer option controls the total cpu utilization of all matching IRQs; this number will be divided evenly across all the matches, and defaults to 0. If you would like to have each IRQ use 100% CPU, and you have 3 matching IRQs, set this to 300. WARNING: this number is multiplied by the `pvtm_agent_cpu_scaling_factor` automatically by the agent so if the `pvtm_agent_cpu_scaling_factor` were 100 in the example above, there's NO NEED to set this value to 30000

The default value is `pvtm_agent_irq_auto_max_total_cpu_util=0`;

`pvtm_agent_irq_files`

This "list of groups" option enables PVTM Agent to inject IRQs artificially in the model. In Windows, this is currently the only way of incorporating IRQs into the system. The IRQs appear as threads with negative PIDs, with connections to threads that have either a socket connection that matches the IRQ's netmask, or an open file that matches the IRQ's file regex.

As listed in the BNF Grammar, the list uses parenthesis with one or more groups of settings. Each of the internal groups is delimited by curly braces `{}`, and is comma-delimited. The internal

groups show information about the interrupts artificially added to the model, and have the following options:

- `irqNum` -this integer option has the interrupt number used by the model to identify this IRQ.
- `allowedCores` - this string option has the list of cores where the model is allowed to place the interrupt. **WARNING:** this option does not use the taskset-style of lists; the cores have to be explicitly set here; also, do not forget that this is a string option, which requires double quotes.
- `currCore` - this integer option has the current core where the IRQ is running. This should match the same value as the `allowedCores` setting.
- `threadPcnts` - this string option has an artificial cpu % that is used by the model when calculating the PVTM scores.
- `regex` - this optional string option has a regular expression that is used by PVTM Agent to create connections to I/O threads accessing matching files
- `cidrs` - this optional string list option has a parenthesis-delimited list of IPv4 address masks in CIDR notation. The CIDR has two parts delimited by a forward slash (/). The first part has an IP address, and the second part the number of ones used in a binary mask that is 'binarily anded' with the IP address. The result is an IP address mask that can be used to form a routing table to match IP addresses.

As an example, the CIDR 192.168.51.0/24 matches any IP addresses that start with 192.168.51.x, whereas 192.168.51.0/16 would match any IP addresses starting with 192.168.x.x. The 192.168.51.0 part of the CIDR is the following in binary:

192	168	51	0
1 1 0 0 0 0 0 0	1 0 1 0 1 0 0 0	0 0 1 1 0 0 1 1	0 0 0 0 0 0 0 0

When anded with the /24 1's bitmask, we have the following:

192	168	51	0
1 1 0 0 0 0 0 0	1 0 1 0 1 0 0 0	0 0 1 1 0 0 1 1	0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0	1 0 1 0 1 0 0 0	0 0 1 1 0 0 1 1	0 0 0 0 0 0 0 0
192	168	51	0

When anded with the /16 1's bitmask, we have the following:

192	168	51	0
1 1 0 0 0 0 0 0	1 0 1 0 1 0 0 0	0 0 1 1 0 0 1 1	0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0	1 0 1 0 1 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
192	168	0	0

Here's a quick example:

```
pvtm_agent_irq_files= (
  {irqNum = 16; allowedCores="[0,2,4,6]"; currCore=0; threadPcnts="[100]";
  regex="/tmp";},
  {irqNum = 15; allowedCores="[0,2]"; currCore=2; threadPcnts="[50]";
  regex="/opt";},
  {irqNum=19; allowedCores="[0,1]"; cidrs=("0.0.0.0/0", "169.254.0.0/16",
  "192.168.51.0/24"); threadPcnts="[0]"; currCore=0;}
);
```

In the example above, PVTM Agent will artificially create 3 IRQs numbered 16, 15, and 19. These IRQs will have the following behaviour:

- 1) IRQ 16 – connect to any threads reading or writing to a file that has the pattern '/tmp; IRQ 16 will be allowed to run on cores 0,2,4,6, will be reported as currently running on core 0, and using 100% CPU.
- 2) IRQ 15 – connect to any threads reading or writing to a file that has the pattern '/opt; IRQ 15 will be allowed to run on cores 0,2,4,6, will be reported as currently running on core 0, and using 100% CPU.
- 3) IRQ 19 – connect to any external IP addresses that match the following criteria:
 - don't match any other NICs (the 0.0.0.0/0 means this is a default gateway),
 - or any IP addresses that start with 192.168.51.x,
 - or any IP addresses that start with 169.254.x.x.

IRQ 19 will be allowed to run on cores 0 and 1, and will be reported as using 0% CPU and running on core 0.

pvtm_agent_irq_route_table_symbols_to_cut

This string option helps PVTM Agent filter out unnecessary entries of the routing table on Linux (this is currently ignored in other platforms). When servers have VLAN tagging on, this can cause issues with the matching. VLAN tags seem to appear as strings with the following format in the routing table: <interface>.<vlan tag>. With the default setting, PVTM Agent cuts out any dots ('.') from the name, starting from the last entry in the table, so entries like the ones below will be saved as just eth0 instead of eth0.102, or eth0.104:

```
cat /proc/net/route
Iface Destination Gateway Flags RefCnt Use Metric Mask MTU Window IRTT
eth0.102 6059E10A 00000000 0001 0 0 0 0 E0FFFFFF 0 0 0
eth4 4059E10A 00000000 0001 0 0 0 0 E0FFFFFF 0 0 0
eth5 4059E10A 00000000 0001 0 0 0 0 E0FFFFFF 0 0 0
eth0.104 6079E10A 00000000 0001 0 0 0 0 E0FFFFFF 0 0 0
eth0.102 00C319AC 6159E10A 0003 0 0 0 0 00FFFFFF 0 0 0
eth0.104 00C419AC 6179E10A 0003 0 0 0 0 00FFFFFF 0 0 0
eth4 0000FEA9 00000000 0001 0 0 1006 0000FFFF 0 0 0
eth5 0000FEA9 00000000 0001 0 0 1007 0000FFFF 0 0 0
eth0.102 0000FEA9 00000000 0001 0 0 1008 0000FFFF 0 0 0
eth0.104 0000FEA9 00000000 0001 0 0 1009 0000FFFF 0 0 0
eth0.104 00000000 6179E10A 0003 0 0 0 0 00000000 0 0 0
```

pvtm_agent_junk_cores

pvtm_agent_junk_cores_json_array

This string option sets the list of junk cores in the machine; if running as root, you can also set the option `pvtm_agent_move_non_match_pids_to_junk_cores` to true to move the non-matching apps to the junk cores. Note that as of version 4.5.0.64, the alternative option name `pvtm_agent_junk_cores` may also be used; however, if both are present in the configuration file, only the value in `pvtm_agent_junk_cores_json_array` will be used. Both options also now allow for mixed list notation (e.g. 1,3,4-7). **Please note that this is a string value with a JSON Array inside, so don't forget the double quotes.**

Junk cores are cores dedicated to non-performance-sensitive applications. Background processes, including some Kernel threads run on junk cores to avoid interference with performance sensitive applications. PVTM Agent can move all non-matching processes, including itself to a list of Junk Cores. It is usually recommended that more than one junk core is used in case an application running in the junk cores uses 100% CPU utilization and renders the server inaccessible.

The default value is `pvtm_agent_junk_cores_json_array="[]"`;

pvtm_agent_junk_cores_allow_threads

This Boolean option determines whether PVTM Agent allows the PVTM Simulator to place threads in the junk cores. This is useful in environments where users need all the cores for the application (e.g. grid environments), but still want to limit the interference caused by non-matching apps to just a few cores. The default value for this option is false, which makes the PVTM Simulator ignore the junk cores during simulations:

```
pvtm_agent_junk_cores_allow_threads = false;
```

pvtm_agent_junk_cores_sanity_check

This Boolean option determines whether PVTM Agent attempts to sanity check the value of the `pvtm_agent_junk_cores` option. If this is set to true (default), PVTM Agent will look at the current set of active CPUs and ensure that the junk cores all fall within the correct range.

The only time when it is worth switching this option to false is on older platforms, where PVTM Agent may not correctly recognize all the cores in the system.

pvtm_agent_link_proto_weights_un_i4_i6_ld_pf_lk_sm_mins_json_array

pvtm_agent_link_proto_weights_un_i4_i6_ld_pf_lk_sm_maxs_json_array

These two string options allow users to bias the weights of inter-thread connections. They contain the min and max values for the normalized weights of the following protocols:

- un - unknown protocol – this is typically a result of an artificially inserted link using the `pvtm_agent_app_link_override` option
- i4 - ipv4 sockets
- i6 - ipv6 sockets
- ld - local domain sockets
- pf - pipes and files
- lk - locks
- sm - shared memory

These arrays will spread the actual values for the protocols (the value in parenthesis of how many unlocks for locks, or bytes were sent on sockets) between these artificial min and max values. The default values below will make all the unknown protocols with a weight of 1, all the ipv4,v6, and local domain sockets spread between 20 and 30, all the pipes and files between 1 and 5, all the locks at 1, and all the shared memory areas at 10. These will effectively bias the model towards favouring socket traffic over locks, pipes and files, and shared memory:

```
pvtm_agent_link_proto_weights_un_i4_i6_ld_pf_lk_sm_mins_json_array="[1,20,20,20,1,1,10]";
```

`pvtm_agent_link_proto_weights_un_i4_i6_ld_pf_lk_sm_maxs_json_array="[1,30,30,30,5,1,10]";`

pvtm_agent_lock_stat_countdown_to_stop_monitoring

This integer option controls how often the PVTM inter-thread discovery mechanism polls for inter-thread locks. When PVTM Agent profiles locks between threads, it has to be as unobtrusive as possible. To minimize the impact on the apps, PVTM Agents will just profile the first 'Lock Stat Countdown' calls every polling period. This option controls the number of unlock calls that are tracked every `pvtm_agent_timer_poll_thread_interval_ms`.

The default value is `pvtm_agent_lock_stat_countdown_to_stop_monitoring=100;`

pvtm_agent_log_enable_syslog

This boolean option controls whether messages from PVTM Agent go into syslog on Linux. The default value is true, which enables syslog messages.

pvtm_agent_log_level

This integer option has a bit mask for the log level; 255 is the full mask. As seen in the table below, 1 is ERROR only, 2 is WARNINGS only, 3 is ERROR and WARNINGS, 4 is NOTICE only, 7 is NOTICE, WARNINGS and ERROR messages:

SAMPLE VALUE	Bit 0 (ERROR)	Bit 1 (WARNING)	Bit 2 (NOTICE)	Bit 3 (INFO)	Bit 4 (DEBUG)	Bit 5 (WebSockets PARSER)	Bit 6 (WebSockets HEADER)	Bit 7 (WebSockets EXT)
255	1	1	1	1	1	1	1	1
7 (Def)	1	1	1	0	0	0	0	0
15	1	1	1	1	0	0	0	0
1	1	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0
10	0	0	0	0	1	0	0	0

The bits 4-7 will likely cause too much data to be logged. It's strongly advised not to do this in a production environment.

The default value is `pvtm_agent_log_level=7;`

pvtm_agent_migrate_memory_when_pinning

This Boolean option determines whether PVTM Agent also moves the target application's memory from one CPU to another. Note that this is a ****VERY EXPENSIVE**** call, and should only be used in environments where there are not a lot of changes in the thread behaviour.

The default value is `pvtm_agent_migrate_memory_when_pinning=false;`

pvtm_agent_min_score_apply_pinning_threshold

This integer option controls the minimum score after which PVTM Agent applies thread pinning strategies. This is useful to prevent wrong pinning strategies to be implemented during quiet periods of execution.

The default value is `pvtm_agent_min_score_apply_pinning_threshold=0;`

pvtm_agent_move_non_match_pids_to_junk_cores

This Boolean option is used to control whether or not to move non-matching apps (not matched by the `pvtm_agent_regex_pattern`) to either the cores listed in `pvtm_agent_junk_cores`, or optionally only in the `pvtm_agent_premium_junk_cores`.

The default value is `pvtm_agent_move_non_match_pids_to_junk_cores=false`;

pvtm_agent_norm_apps_send_current_cores

This Boolean option controls whether PVTM Agent sends the current core location of each of the threads when requesting a normalized simulation request. This is useful when the current pinning position needs to be used as a reference to reduce the number of changes between the current state and the target-simulated state.

The default value for this option is `pvtm_agent_norm_apps_send_current_cores = true`;

pvtm_agent_num_times_no_significant_changes_force_send

This integer option controls the number of times that no significant changes in CPU happen before a new score request is forced. PVTM Agent sends `GET_SCORE` requests whenever the CPU utilization of any of the threads changes by a configurable percentage threshold, whenever a new thread starts, or PVTM Agent can also be configured to force a score request to be sent periodically regardless of any changes in the environment by setting this option.

The default value for this option is `pvtm_agent_num_times_no_significant_changes_force_send=5`;

pvtm_agent_passive_pinning_mode

This Boolean option prevents PVTM Agent from applying the thread pinning strategies, but the requests and thread pinning responses can still go to/from the thread manager if the `pvtm_agent_passive_pinning_mode_send_requests` option is set to true. Change this option to false to start applying the pinning strategies and sending requests.

The default value is `pvtm_agent_passive_pinning_mode=true`;

pvtm_agent_passive_pinning_mode_send_requests

This Boolean option controls whether PVTM Agent sends requests when the option `pvtm_agent_passive_pinning_mode` is set to true. Setting this option to true whilst in passive mode is useful for building a cache of system states and gauging how good is the model. If this option is set to false and the `pvtm_agent_passive_pinning_mode` is set to true, then no run requests go to the PVTM Simulator. This option is ignored when PVTM Agent is in active mode.

The default value is `pvtm_agent_passive_pinning_mode_send_requests=false`;

pvtm_agent_premium_junk_cores

This string option sets an optional list of 'premium' junk cores that can be used for running other apps (note that the `pvtm_agent_premium_junk_cores` option only has an impact if the `pvtm_agent_move_non_match_pids_to_junk_cores` option is set to true).

WARNING: this option has to have a subset of cores to the ones on the `pvtm_agent_junk_cores_json_array`. If the set of cores here is not a subset, then the `pvtm-agent` will exit.

The Premium Junk Cores concept allows users to choose a sub-set of cores to run non-matching applications within the junk cores. This is particularly useful if hyperthreading is turned on in a server, and a user wants to logically disable them. This can be done by setting the list of junk cores to all the hyper threads, plus a couple of real junk cores, and then setting the premium junk cores to just the real junk cores. This would prevent the simulator from using any hyper threads or the normal junk cores, and it would force PVTM Agent not to run any apps in the hyperthreads.

As seen above, this option was created to allow PVTM Agent to mask out hyperthreads in a server, whilst still getting the freedom to constrain non-matching apps to a sub-set of the cores. This option allows for mixed list notation (e.g. 1,3,4-7) with or without the square brackets; however, **the value must be a string, so please don't forget the double quotes** (e.g. `pvtm_agent_premium_junk_cores="[1,3-7]";`).

If this option is not set, the default value is the same as `pvtm_agent_junk_cores_json_array`.

pvtm_agent_process_conn_stats

This Boolean option determines whether PVTM Agent processes sockets, files and named pipe statistics. When set to true, any such statistics found by PVTM Agent will be used. If set to false, all the sockets, files, and named pipe statistics will be ignored.

The default value is `pvtm_agent_process_conn_stats=true;`

pvtm_agent_process_lock_stats

This Boolean option determines whether PVTM Agent processes futex lock statistics. When set to true, any lock statistics found by PVTM Agent will be used. If set to false, all the lock statistics will be ignored.

The default value is `pvtm_agent_process_lock_stats=true;`

pvtm_agent_regex_pattern

This string option is one of the most important in PVTM Agent, as it enables it to decide which applications are considered performance-sensitive, and which do not. The value of this option is a regular expression that controls which apps PVTM Agent will match. The regex match is used in all the GUI representations of the system, so care should be taken not to make it too generic. For example, if a system has all java-based applications, using a regex of `java` is quite easy to configure; however, it is not very descriptive. If, however, the description of the app appeared somewhere in the command line such as `-Dapp_description="app1"`, then it would make more sense to use the word after `app_description` instead of `java`. This could be done using the following regex: `"(?=.*Dapp_description=(\\w+))"`. The downside of this approach is that the

regex is harder to create, and that PVTM Agent has to do additional work to match command line arguments, so if there are large numbers of processes running, this can make an impact in performance.

The default value is `pvtm_agent_regex_pattern="java"`;

Here are some other examples:

```
//pvtm_agent_regex_pattern="(?.*switched.(\w+))";
//pvtm_agent_regex_pattern=".*";
//pvtm_agent_regex_pattern="(=?parallel (\w+)|sequential (\w+))";
//pvtm_agent_regex_pattern="net...";
//pvtm_agent_regex_pattern="EXCEL";
pvtm_agent_regex_pattern="java";
//pvtm_agent_regex_pattern="notepad$";
```

pvtm_agent_regex_pattern_results_delimiter

This string option controls whether PVTM Agent appends a string in between multiple regex results in the same match. For example, if a process name "foobarfoo.x" is matched with the `pvtm_agent_regex_pattern "foo"`, and the `pvtm_agent_regex_pattern_results_delimiter` is set to "__", then the result that will appear on the PVTM GUI is `foo__foo`.

The default value is `pvtm_agent_regex_pattern_results_delimiter=""`;

pvtm_agent_run_config_apply_pinning_per_thread

This Boolean option controls whether the individual threads get pinned to just their own cores. If setting this to true, each individual thread will be given a mask. This is useful in applications that have long-lived thread pools.

The default value is `pvtm_agent_run_config_apply_pinning_per_thread=true`;

pvtm_agent_run_config_apply_pinning_per_thread_merge_last_n_solutions

This integer option allows PVTM Agent to set the affinity of individual threads not to only the latest simulation result's core, but to a set of the last N results (N being the value of this option). Note that only the last 16 results are currently stored. As example, let us use a hypothetical application has two threads, and the following previous simulation results:

```
Time 0 – Thread 0 – core 0, Thread 1 – core 10
Time 1 – Thread 0 – core 5, Thread 1 – core 10
Time 2 – Thread 0 – core 7, Thread 1 – core 11
Time 3 – Thread 0 – core 10, Thread 1 – core 11
```

If the latest thread pinning strategy happened at Time 3, then here are some scenarios:

- If the value of this option is 1, then Thread 0 will be pinned to cores 7 and 10, and Thread 1 to core 11.
- If the value of this option is 3, then Thread 0 will be pinned to cores 0,5,7,10, and Thread 1 to cores 10 and 11.
- Lastly, if this option is set to 4 and above, then Thread 0 will be pinned to cores 0,5,7,10, and Thread 1 to cores 10 and 11.

This option helps improve thread pinning of environments with more active threads than cores; by giving the scheduler a few options in busy environments, it allows the system to react to quick changing environments.

The default value of zero does not attempt to do any merging.

The default value is the following:

```
pvtm_agent_run_config_apply_pinning_per_thread_merge_last_n_solutions=0
```

This only runs the last simulation results.

pvtm_agent_run_config_apply_pinning_per_thread_merge_last_n_solutions_cpu_threshold

This integer option controls the CPU percentage (multiplied by the cpu scaling factor), after which no merging will occur. If this value is set to 50, and the scaling factor is 1, then any threads that have more than 50% CPU percentage utilization will not have the last_n_solutions merged; n will be forced to be 0.

The default value for this option is 0:

```
pvtm_agent_run_config_apply_pinning_per_thread_merge_last_n_solutions_cpu_threshold=0;
```

pvtm_agent_run_config_apply_pinning_whole_process

This Boolean option controls whether the whole app gets pinned to all the cores or just the individual threads get pinned to just their own cores. If setting this to true, the whole process will be given a mask. This is useful in applications that have dynamic thread pools that are short-lived. Note that this should always set to true in Windows.

The default value is `pvtm_agent_run_config_apply_pinning_whole_process=true;`

pvtm_agent_run_solution_cache_app_thread_pct_only

This Boolean option enables PVTM Agent to use the thread percentages (rather than the whole state of the applications) as a hash key for previous simulation results. This may lead to a higher cache ratio at the risk of producing less accurate pinning strategies. .

The default value is `pvtm_agent_run_solution_cache_app_thread_pct_only =false;`

pvtm_agent_run_solution_cache_enabled

This Boolean option enables the caching capabilities in the PVTM Agent. Users are strongly advised not to change this option to false, as it could lead to no matches of previous simulations being found, and increased CPU Utilization in PVTM Simulator.

The default value is `pvtm_agent_run_solution_cache_enabled=true;`

pvtm_agent_run_solution_cache_lru_table_size

This integer option controls the number of previous simulation results the agent will cache locally in memory. Users are strongly advised not to set this value too low, as it could lead to no matches of previous simulations being found, and increased CPU Utilization in PVTM Simulator.

The default value is `pvtm_agent_run_solution_cache_lru_table_size=100`;

pvtm_agent_run_solution_command

This string option controls the various different algorithm parameters to control the Simulator when `RUN_SOLUTION_CACHED` request messages are sent. The values used here can be experimented during off-line Simulations using the PVTM GUI. The value of option is a string with a JSON-like object inside, including the following parameters:

- 'inertiaCpuUtilThreshold' – a single-quoted string representing the CPU utilization where the PVTM Simulator starts applying an inertia penalty. The inertia penalty penalizes threads that have high CPU utilizations (above the `inertiaCpuUtilThreshold`) and that move away from the existing position. If not set, the default value is 100, which means that no threads will have inertia applied to them (as no threads can go over 100% CPU utilization). This value ranges from 0 to 100.
- 'inertiaMultiplier' – a single-quoted string representing the double floating point precision values that PVTM Simulator multiplies by the inertia penalty. The inertia penalty penalizes threads that have high CPU utilizations (above the `inertiaCpuUtilThreshold`) and that move away from the existing position. If not set, the default value is 0, which means that no threads will have inertia applied to them. This value should be a positive floating point number. If this value is greater than 1.0, it exacerbates the impact of inertia (e.g. ensure that high CPU threads never get moved). If this value is between 0 and 1, it can alleviate the inertia penalty (e.g. give the model a bit more freedom when moving threads with more than 1% CPU).
- `cpuScalingFactor` is a dividing factor used to obtain CPU utilizations that are less than one percent. For efficiency reasons, the CPU utilization is always listed as an integer value. If an application requires sub 1% CPU accuracy, then this scaling factor is used by PVTM Agent to multiply the CPU counters by this value when calculating its percentages, so if the `cpuScalingFactor` is 10, then a process using 0.5 % CPU will be reported as 5, whereas a process using 100% CPU will be reported as 1000. The simulator then uses this scaling factor to re-normalize the CPU utilization when the context switch penalties are calculated.
- 'maximumSecondsSpend' – a single-quoted string representing the maximum number of seconds the PVTM Simulator will spend running a simulation; the default value of 30 seconds should be more than enough for most environments, and if the system has a relatively small number of solutions, this value can be set to 10 seconds or lower.
- 'randomize' – single quoted string with a Boolean value representing whether or not PVTM Simulator should randomize the initial thread pinning assignment. This can have a positive impact in the quality of the results in some systems, but may also have a negative impact in others.
- 'clearLast' – single quoted string with a Boolean value representing whether or not PVTM Simulator should clear the current thread execution layout, and use one of the 'constructionHeuristicType' algorithms instead. Setting this to false will cause the model to use the current thread execution layout as a starting point. If absent, this defaults to false, and the current thread execution layout is used as a starting point; this usually causes the least number of thread changes, but may lead to local optima.

- 'constructionHeuristicType' single quoted string with an algorithm used by the simulator to place the initial set of threads if the 'clearLast' value is set to 'true'. Here are the possible values:
 - FIRST_FIT – the PVTM Simulator uses the first valid (i.e. that only uses valid cores) first thread execution layout it can find in increasing order of complexity, placing lower percentage threads first, then threads with fewer neighbour connections and then continuing from there regardless of scores.
 - FIRST_FIT_DECREASING – the same as first fit, but in reverse order.
 - WEAKEST_FIT – The same as FIRST FIT, but keeping scores, and using the ****worst**** scores first.
 - WEAKEST_FIT_DECREASING – The same as FIRST_FIT_DECREASING, but keeping scores, and using the ****worst**** scores first.
 - STRONGEST_FIT – The same as FIRST FIT, but keeping scores, and using the ****best**** scores first.
 - STRONGEST_FIT_DECREASING – The same as FIRST FIT, but keeping scores, and using the ****best**** scores first.
 - ALLOCATE_ENTITY_FROM_QUEUE – Experimental algorithm; please do not use this.
 - CHEAPEST_INSERTION – Experimental algorithm; please do not use this.
 - ALLOCATE_FROM_POOL – Experimental algorithm; please do not use this.
- 'secondarySearchAlgos' – after insertion, the algorithm used to improve the score. It's strongly recommended that you contact Pontus's professional services to change this value. For completeness, here is the list of possible values:
 - ENTITY_TABU (Default, with a 2% TABU, and no, this is not taboo misspelled, it is indeed TABU – see https://en.wikipedia.org/wiki/Tabu_search for more details).
 - SIMULATED_ANNEALING (see https://en.wikipedia.org/wiki/Simulated_annealing for more details)
 - LATE_ACCEPTANCE (see <http://www.cs.stir.ac.uk/research/publications/techreps/pdf/TR192.pdf> for more details)
 - STEP_COUNTING_HILL_CLIMBING
 - HILL_CLIMBING (brute force approach trying every possible combination – this is not feasible for large search spaces)
 - ENTITY_TABU,MOVE_TABU
 - ENTITY_TABU,UNDO_MOVE_TABU
 - SIMULATED_ANNEALING,ENTITY_TABU
 - SIMULATED_ANNEALING,ENTITY_TABU,MOVE_TABU
 - LATE_ACCEPTANCE,MOVE_TABU

Here is a sample setting for this value:

```
pvtm_agent_run_solution_command="{
  'maximumSecondsSpend':'10','randomize':'true','constructionHeuristicType':'S
  TRONGEST_FIT_DECREASING','secondarySearchAlgos':'HILL_CLIMBING' }"
```

Here is another example that sets the inertiaCpuUtil threshold to a value of 90% :

```
pvtm_agent_run_solution_command="{
  'maximumSecondsSpend':'30','randomize':'false','inertiaCpuUtilThreshold':'90'
  }"
```

pvtm_agent_run_solution_command_after_new_process_delay

This string option controls the command parameters sent to the simulator (using the same semantics as the `pvtm_agent_run_solution_command` option) after PVTM Agent senses the start of a new application that matches the `pvtm_agent_regex_pattern`. Once a new application starts, PVTM Agent waits `pvtm_agent_run_solution_new_process_delay_ms` milliseconds before using this alternative set of commands for any future thread pinning requests. If this option is set, PVTM Agent only uses commands in the option `pvtm_agent_run_solution_command` between the start of a new matching application and the expiry of the number of milliseconds set in the `pvtm_agent_run_solution_new_process_delay_ms` option.

The default value is for this option to be unset.

This option is very useful for modifying the PVTM Simulator behaviour slightly after new processes arrive. One useful use case for this is to set the 'inertiaCpuUtilThreshold' value to a value lower than '100' slightly after a new application starts. This enables the PVTM Simulator model to freely move threads as soon as a process starts, and then, after a warm-up period, add inertia so the busier threads stop moving.

pvtm_agent_run_solution_new_process_delay_ms

This integer option controls the delay (in milliseconds) before PVTM Agent starts using the `pvtm_agent_run_solution_command_after_new_process_delay` commands for simulation requests. The delay always starts after PVTM Agent senses the start of a new application that matches the `pvtm_agent_regex_pattern`. Between the start time and this delay, PVTM Agent uses the commands in the `pvtm_agent_run_solution_command` to request simulations from the PVTM Simulator. The default value for this option is 30000 (30 seconds): `pvtm_agent_run_solution_new_process_delay_ms=30`; note that this option only takes effect if the option `pvtm_agent_run_solution_command_after_new_process_delay` is set.

pvtm_agent_same_core_cost

This integer option controls the cost of running apps on the same core. Increasing this value will artificially inflate the cost of running apps that communicate with each other in the same core. You should increase this value if the model is trying to run too many apps in the same core. The default (-1) uses the actual value reported by the OS.

The default value if this option is unset is -1; however, the recommended value is `pvtm_agent_same_core_cost=40`;

pvtm_agent_score_callback_lib_file

This string option has the name of a shared library (or DLL) that has alternative implementations of the `defaultNeedToRunSolutionCb()` function (see Customizing When to Run a Solution Request with `defaultNeedToRunSolutionCb()` for more details).

The default value is an empty string, which causes PVTM Agent to use the default decision-making to determine when a thread pinning strategy should be requested.

pvtm_agent_score_callback_function_name

This string option has the name of a function inside the `pvtm_agent_score_callback_lib_file` library (or DLL) that has alternative implementations of the `defaultNeedToRunSolutionCb()` function (see

Customizing When to Run a Solution Request with `defaultNeedToRunSolutionCb()` for more details).

The default value is an empty string, which causes PVTM Agent to use the default decision-making to determine when a thread pinning strategy should be requested.

`pvtm_agent_score_moving_average_backoff_if_too_high`

This Boolean Option controls whether or not PVTM Agent should reset the moving average counter if a spike in the score is greater than `pvtm_agent_score_moving_average_period * pvtm_agent_delta_score_threshold_percentage / 100`.

The default value is `pvtm_agent_score_moving_average_backoff_if_too_high = false`;

`pvtm_agent_score_moving_average_period`

This integer option controls the number of previous scores to be used in a moving average of scores. PVTM Agent keeps a rolling simple moving average of the last 'Moving Average Period' scores, and uses this number to determine whether a new thread pinning strategy is required.

Note that after re-starting PVTM Agent, an automatic `RUN_SOLUTION_CACHED` request will be sent as soon as the moving average vector is filled, so this value can also delay or shorten the time of the first thread pinning run solution.

The default value is `pvtm_agent_score_moving_average_period=5`; this value should always be greater than 1.

`pvtm_agent_score_num_ignore_after_pinning`

This integer option makes PVTM Agent ignore the next `n` scores after a thread pinning strategy was applied. This helps reduce the number of unnecessary pinning strategies, giving the system a few seconds to settle down after being re-pinned. Note that the number of scores set in this option will not be added to the moving average after a thread execution layout was implemented.

The default value for this option is `pvtm_agent_score_num_ignore_after_pinning=3`;

`pvtm_agent_search_command_line`

This Boolean option controls whether or not the command line arguments are also searched for the regex set in the option `pvtm_agent_regex_pattern`. By default, only the executable name is searched. Where possible, this should be set to false, as setting it to true, especially if there are many running processes in the system, may cause increased CPU Utilization in the PVTM Agent.

The default value for this option is `pvtm_agent_search_command_line=false`;

`pvtm_agent_search_env_vars`

This Boolean option Boolean controls whether or not the environment variables of all processes are also searched for the regex set in the option `pvtm_agent_regex_pattern`. By default, only the executable name is searched. Where possible, this should be set to false, as setting it to true, especially if there are many running processes in the system, may cause increased CPU Utilization in the PVTM Agent.

The default value for this option is `pvtm_agent_search_env_vars=false`;

pvtm_agent_send_thread_names

This Boolean option controls whether or not PVTM Agent sends thread names to PVTM Simulator. This option currently only impacts Linux. Whilst this option makes it easier to view information in the PVTM GUI, it does not significantly impact the PVTM Agent functionality; for applications with large numbers of threads, this option may also significantly increase the bandwidth and disk utilization in the PVTM Simulator.

The default value `pvtm_agent_send_thread_names = false`;

pvtm_agent_set_chrt_rr_priority

This integer option controls the priority level to which PVTM Agent changes any matching threads. This option will be ignored unless the `pvtm_agent_chrt_rr_matching_processes` option is set to true.

The default value for this option is 97 on Linux, and -3 on Windows

WARNING: you must run PVTM Agent as root for this to work.

Notice: On Linux the value in here will appear in top as a negative value.

pvtm_agent_store_last_core_on_command_force_run_solution_cached

This Boolean option controls whether PVTM Agent stores the last core where each thread was running when a `COMMAND_FORCE_RUN_SOLUTION_CACHED:SIZE=<size>:<data>` message arrives. This is used by the LD_PRELOAD library on linux when the environment variable `PVTM_ENABLE_SELF_PINNING` is set.

pvtm_agent_timer_poll_cfg_file_interval_ms

This integer option controls how often (in milliseconds) PVTM Agent looks for dynamic configuration file changes. PVTM Agent can periodically poll a dynamic configuration file to look for any dynamic options. This is a mechanism to change rules of artificial inter-thread connections without stopping and starting PVTM Agent. This option sets the number of milliseconds between each poll. Notice that the file is only processed if its access timestamp has been modified. This option has no impact unless the `pvtm_agent_dynamic_config_file_name` is set.

The default value is `pvtm_agent_timer_poll_cfg_file_interval_ms=30000`;

pvtm_agent_timer_poll_command_req_interval_ms

This integer option controls how often (in milliseconds) PVTM Agent looks for new Command and Control requests. If no requests are outstanding, this mechanism behaves like a heartbeat. Setting this value to zero or a negative number disables this feature.

The default value is `pvtm_agent_timer_poll_command_req_interval_ms=0` (Disabled). If enabling this feature is required, the recommended value is 5000.

pvtm_agent_timer_poll_proc_interval_ms

This integer option controls how often (in milliseconds) PVTM Agent looks for new processes matching the regular expression set in [pvtm_agent_regex_pattern](#).

pvtm_agent_timer_poll_proc_interval_ms=15000;

pvtm_agent_timer_poll_thread_interval_ms

This integer option controls how often (in milliseconds) PVTM Agent looks for new threads, and how often PVTM Agent measures the CPU utilization of existing threads. Once PVTM Agent finds a list of processes that match the regex filter, it will start introspection of the threads for their CPU utilization. The Timer Thread Interval controls how often this happens. Making this value too long may makes the thread manager lose too many thread patterns; similarly, making this value too short may impact the performance of PVTM Agent, and make it too reactive to small changes in application behaviour. This value must always be lower than the pvtm_agent_timer_poll_proc_interval_ms value.

The default value is pvtm_agent_timer_poll_thread_interval_ms=2000;

pvtm_agent_unsolicited_run_threshold

This integer option controls how often to send unsolicited run solution requests to PVTM Simulator. Users can configure PVTM Agent to send optionally a RUN_SOLUTION_CACHED request message not triggered by any changes in scores. The pvtm_agent_unsolicited_run_threshold option controls how often these unsolicited requests occur. To disable this feature, set this option to -1.

The default value is pvtm_agent_unsolicited_run_threshold=50;

pvtm_agent_wait_before_sending_run_solution_ms

This integer option makes PVTM Agent wait pvtm_agent_wait_before_sending_run_solution_ms milliseconds from the time the last STOP_SOLUTION request message was sent. This gives the agent a cool-off period preventing it from over pinning the environment.

The default value is pvtm_agent_wait_before_sending_run_solution_ms = 10000;

pvtm_agent_wait_outstanding_pinning_request

This Boolean option causes PVTM Agent to wait for a solution result to return before sending the next RUN_SOLUTION_CACHED request message.

The default value is pvtm_agent_wait_outstanding_pinning_request = false;

pvtm_agent_web_socket_interval_us

This integer option controls how often (in microseconds) PVTM Agent waits for new data from each web socket in use. Setting this value too low may cause increased CPU utilization. Note that this value will impact the minimum frequency of the pvtm_agent_timer_poll_thread_interval_ms. Users should not normally change this, unless the pvtm_agent_timer_poll_thread_interval_ms is not matching the requested period. As an example, this option should not be changed, if setting the pvtm_agent_timer_poll_thread_interval_ms option to 300ms, and waiting for data on each

web socket for that period will cause longer delays than 300ms. This happens because PVTM Agent will go through each of its sockets and wait that long.
The default value is `pvtm_agent_web_socket_interval_us=300000`;

pvtm_agent_web_socket_max_payload_bytes

This integer option controls the maximum packet size allowed to be sent by the agent to the server. In large servers with large number of threads, or large numbers of inter-thread connections, the size of the messages to PVTM Simulator can grow quite large. If the messages are larger than the value in this option, PVTM Agent will not be able to send messages to PVTM Simulator. This usually manifests itself as a socket error message that keeps on appearing on PVTM Agent's console. In those cases, it may be worth increasing this value.

WARNING: the agent's memory footprint will increase 10 x this value, as we currently have // 10 web sockets conns per client.

NOTE: if you do see socket errors, a better option to also change the following option in the `run-threadmgr.(sh|bat)` file:

`-Dpvtm.websockets.maxTextMessageBufferSize=8600000`

The default value for this option is `pvtm_agent_web_socket_max_payload_bytes=2097152`;

Chapter 6. PVTM Simulator

PVTM Simulator is a 100% Java application that typically runs on a separate server to the one that needs to be optimized, and performs three main functions:

1. PVTM Score Calculation – computing a PVTM Score that reflects the current state of a system,
2. Thread Execution Layout Simulation - running millions of simulations to optimize the score
3. Historical Repository - storing any significant changes in the environment, including previous simulation results in an elastic search big-data database.

The following sections will cover each of these three activities in more detail.

PVTM Score Calculation

PVTM Score calculation is one of the most important features of Pontus Vision Thread Manager (PVTM). PVTM Simulator penalizes poor system behaviour by adding values to the PVTM Score; the lower the PVTM Score, the better the application performance should be. The PVTM Score of any running system should always be a positive integer greater than zero. A zero score typically means that the PVTM Agent did not find any applications that matched the criteria, and as such is invalid. Negative scores are not allowed. Other instances where a score appears as positive, but may still be invalid happen when threads (or IRQs) are running in a list of cores that is either not allowed for their parent process, or that are running in a list of junk cores.

The PVTM Score combines two main factors that determine how well the system should perform:

- Context Switches, or, in PVTM Simulator's terms, the number of active threads that are competing for the same core
- Inter-thread Communication Costs, or, in PVTM Simulator's terms, the cost of moving data between threads

The following sections cover the algorithms used to calculate these factors.

Context Switches

As seen in the `pvtm_agent_context_switch_cost` section, the simulator penalizes context switches by using the following formula:

$$Pct = \frac{\ln(numThreads) * CPUUtil * ContextSwitchCost}{CpuScalingFactor}$$

Where:

`numThreads` is the number of threads currently scheduled in the core,

`CPUUtil` is the total CPU percentage of the threads currently scheduled in the core, or 0.01 if the total util is zero)

`ContextSwitchCost` is the value set in the `SET_COMPUTER` message

`CPUScalingFactor` is either 1, or the value set in the `RUN_SOLUTION_CACHED` request message

This formula penalizes a large number of threads running on a set of cores exponentially, and if the number of threads is zero, a zero penalty is applied. For example, let's assume that a system has 4 cores, and 10 threads running with the following layout shown in [Figure 5](#), with a context switch cost value of 1000, and a CPU Scaling factor of 1:

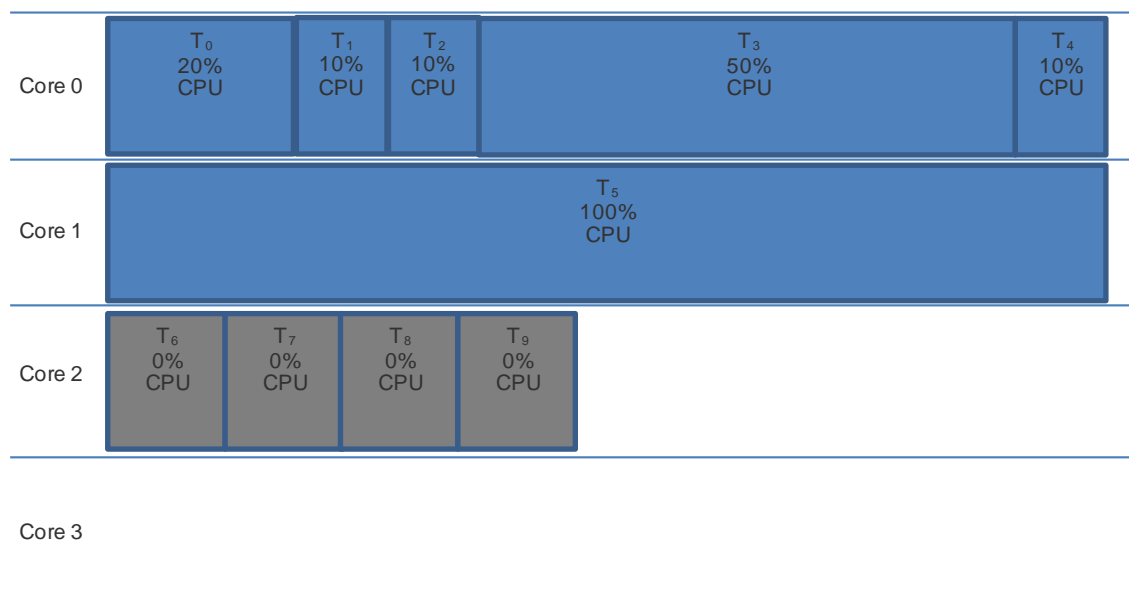


Figure 5 – Sample Thread Execution Layout

In its current state, the context switch penalty score for the system would be approximately 160958; here's the breakdown of applying the formula for each core:

$$\begin{aligned}
 \text{Core 0 Penalty} &= \ln(5) * 100 * 1000 = 160944 \\
 \text{Core 1 Penalty} &= \ln(1) * 100 * 1000 = 0 \\
 \text{Core 2 Penalty} &= \ln(4) * 0.01 * 1000 = 14 \\
 \text{Core 3 Penalty} &= = 0
 \end{aligned}$$

In contrast, if we re-arranged the threads to be more equally divided as in [Figure 6](#), the context switch penalty score for the system would be approximately 56630; here is the breakdown of applying the formula for each core:

$$\begin{aligned}
 \text{Core 0 Penalty} &= \ln(6) * 20 * 1000 = 35835 \\
 \text{Core 1 Penalty} &= \ln(2) * 30 * 1000 = 20794 \\
 \text{Core 2 Penalty} &= \ln(1) * 100 * 1000 = 0 \\
 \text{Core 3 Penalty} &= \ln(1) * 50 * 1000 = 0
 \end{aligned}$$

Notice that the layout in [Figure 6](#) is likely to cause fewer context switches to the busier threads (T₃ and T₅) by leaving them without any other contention. At the same time, threads T₀ and T₄, with a total combined CPU utilization of 30%, only have each other to contend. Lastly, threads T₁ and T₂ have all the other zero percent threads T₆₋₉ (which may periodically wake up) to contend with.

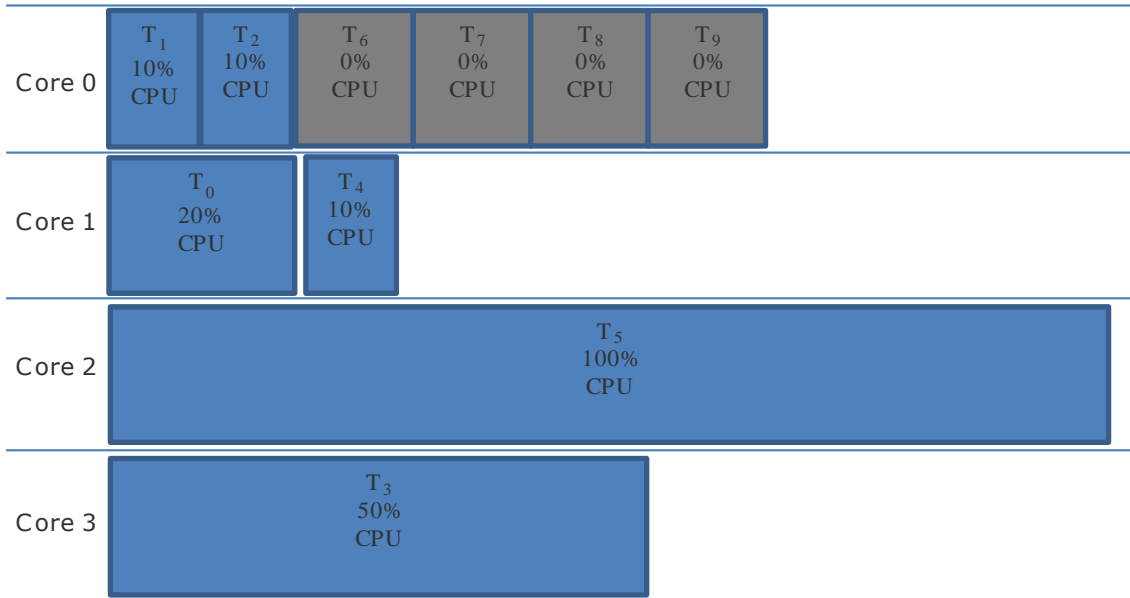


Figure 6 - Re-balanced thread execution layout

It is also important to notice that not every application needs to be penalized by the context switch costs in the same manner. For example if the goal of the simulation were to reduce the number of active threads, and run the same Thread Execution layout as [Figure 5](#), the contextSwitchCost parameter could be set to 0. In practise, however, the extra length of time needed to run the application would likely suffer dramatically from doing that, negating any power savings achieved by not using the 2 cores, thus it is not recommended to set the contextSwitchCost to 0.

Inter-thread Communication Cost

The second main factor involved in the score is the inter-thread communication cost. To calculate this cost, PVTM Simulator uses the following formula:

$$P_{itcc} = \sum_{k=0}^n (\text{cost}(k_{core}, n_{core}) \cdot \text{connWeight}(k, n))$$

Where:

- k is the current thread, n is the number of neighbours to k,
- cost (k_{core}, n_{core}) is the cost between the current core where thread k is located and the current neighbour n,
- connWeight(k,n) is the weight of the connection between thread k and the current neighbour n.

If a thread has no neighbours, then a penalty of 1 is applied. The details of how the costs are obtained are beyond the scope of this document. The following PVTM Agent options lead to an increase in the score, as they either impact the weights between connections, or artificially add new connections:

- pvtm_agent_link_proto_weights_un_i4_i6_ld_pf_lk_sm_mins_json_array
- pvtm_agent_link_proto_weights_un_i4_i6_ld_pf_lk_sm_maxs_json_array

Thread Execution Layout Simulation

The thread execution layout simulation is the PVTM Simulator activity that finds an optimized solution to any given Target Software Model running in a Target Hardware Model. During the thread execution layout simulation, PVTM Simulator selectively moves application threads to different cores continuously re-calculating the PVTM score after each move. The thread execution layout simulation has two main phases: initial placement, and score improvement.

Initial Placement

During the initial placement phase, PVTM Simulator can use one of three strategies:

1. Use the current thread execution layout as the starting point (default in the PVTM Agent)\
2. Randomly assign threads to cores without allowing any invalid thread allocations (e.g. threads placed outside the allowed cores, or threads placed in junk cores).
3. Clear the current thread layout, and apply one of the following mechanisms:
 - a. `FIRST_FIT` – the PVTM Simulator uses the first valid (i.e. that only uses valid cores) first thread execution layout it can find in increasing order of complexity, placing lower percentage threads first, then threads with fewer neighbour connections and then continuing from there regardless of scores.
 - b. `FIRST_FIT_DECREASING` – the same as first fit, but in reverse order.
 - c. `WEAKEST_FIT` – The same as `FIRST_FIT`, but keeping scores, and using the **worst** scores first.
 - d. `WEAKEST_FIT_DECREASING` – The same as `FIRST_FIT_DECREASING`, but keeping scores, and using the **worst** scores first.
 - e. `STRONGEST_FIT` – The same as `FIRST_FIT`, but keeping scores, and using the **best** scores first.
 - f. `STRONGEST_FIT_DECREASING` – The same as `FIRST_FIT`, but keeping scores, and using the **best** scores first.
 - g. `ALLOCATE_ENTITY_FROM_QUEUE` – Experimental algorithm; please do not use this.
 - h. `CHEAPEST_INSERTION` – Experimental algorithm; please do not use this.
 - i. `ALLOCATE_FROM_POOL` – Experimental algorithm; please do not use this.

Score Improvement

After the initial placement, PVTM Simulator tries millions of thread moves to improve the score. PVTM Simulator can use a number of algorithms to organize and implement these moves. The default algorithm is versatile enough to cope with most scenarios. It is strongly recommended to contact Pontus's professional services to get help to change the algorithm. A deep explanation for what these algorithms do is beyond the scope of this document. For completeness, here is the list of the current choices:

- `ENTITY_TABU` (Default, with a 2% TABU, and no, this is not taboo misspelled, it is indeed TABU – see https://en.wikipedia.org/wiki/Tabu_search for more details).
- `SIMULATED_ANNEALING` (see https://en.wikipedia.org/wiki/Simulated_annealing for more details)
- `LATE_ACCEPTANCE` (see <http://www.cs.stir.ac.uk/research/publications/techreps/pdf/TR192.pdf> for more details)
- `STEP_COUNTING_HILL_CLIMBING`

- HILL_CLIMBING (brute force approach trying every possible combination – this is not feasible for large search spaces)
- ENTITY_TABU,MOVE_TABU
- ENTITY_TABU,UNDO_MOVE_TABU
- SIMULATED_ANNEALING,ENTITY_TABU
- SIMULATED_ANNEALING,ENTITY_TABU,MOVE_TABU
- LATE_ACCEPTANCE,MOVE_TABU

Historical Repository

PVTM Simulator uses an embedded Elastic Search 2.0 big data database to store any significant changes in the environment, including previous simulation results. The following sections cover the layout of the database, as well as a few useful queries.

PVTM Elastic Search Schema

PVTM Simulator keeps each PVTM Agent's set of messages stored in a different index with the PVTM Agent's name (usually the host name in the machine being optimized).

The following sections show the mappings that PVTM Simulator keeps for each index:

"RUN_SOLUTION_CHECK_CACHED"

This mapping stores previously cached simulation results. It has the following properties:

- "normReqHash" : a string hash computed by PVTM Agent to match previous simulation results
- "normResults" : a complex object with the normalized simulation results; it has the following properties of its own:
 - a. coreIds – a long property with the last core number where a thread was running
 - b. numaZoneIds – a long property with the last numaZoneId where a thread was running
 - c. pid – a long property with the process ID
 - d. threadIds: a long property with the thread ID
- "score" : a long property with the score computed for this set of simulation results
- "time" : a long property with an epoch timestamp (number of seconds since Jan 1st 1970)

"RUN_SOLUTION_CHECK_CACHED_REQUEST"

This mapping stores simulation requests made by PVTM Agent. It has the following properties:

- "normReq" a complex object with the normalized request from PVTM agent. It has the following properties of its own:
 - a. "computerJson" : a JSON representation of the SET_COMPUTER message sent by PVTM Agent

- b. "normAppLinksJson" : a JSON representation of the normalized SET_APPLINKS message sent by PVTM Agent (normalized in this context means stripped of any process names or process IDs).
 - c. "normAppsJson" : a JSON representation of the normalized SET_APPS message sent by PVTM Agent.
- "normReqHash" : a string hash computed by PVTM Agent to match previous simulation results
 - "reason" : a string property describing the reason why the request was originally made (this is used by PVTM GUI to highlight the green dots in the time series graphs.
 - "score" : a long property with the score of the system at the time the request was made.
 - "time" : a long property with an epoch timestamp (number of seconds since Jan 1st 1970)

"SET_COMPUTER"

This mapping is stored whenever PVTM Agent sends a new SET_COMPUTER message to PVTM Simulator. It has two main properties:

- "computer" : a string property with a JSON representation of the SET_COMPUTER message sent by PVTM Agent
- "time" : a long property with an epoch timestamp (number of seconds since Jan 1st 1970)

"SET_APPLINKS"

Similar to "SET_COMPUTER", this mapping is stored whenever PVTM Agent sends a new SET_APPLINKS message to PVTM Simulator. It has three main properties:

- "appLinks" : a string property with a JSON representation of the SET_APPLINKS message sent by PVTM Agent
- "score" : a long property with the score of the system at the time the request was made.
- "time" : a long property with an epoch timestamp (number of seconds since Jan 1st 1970)

"SET_APPS"

Similar to "SET_COMPUTER", this mapping is stored whenever PVTM Agent sends a new SET_APPS message to PVTM Simulator. It has three main properties:

- "appLinks" : a string property with a JSON representation of the SET_APPS message sent by PVTM Agent
- "score" : a long property with the score of the system at the time the request was made.
- "time" : a long property with an epoch timestamp (number of seconds since Jan 1st 1970)

"RUN_SOLUTION_STATUS"

This mapping stores status messages coming from PVTM Agent whenever a pinning strategy was implemented. PVTM GUI interprets this mapping as red dots in the time series graphs.

The mapping has three main properties:

- "normReqHash" : a string hash computed by PVTM Agent to match previous simulation results
- "score" : a long property with the score of the system at the time the request was made.

- "time" : a long property with an epoch timestamp (number of seconds since Jan 1st 1970)

"EVENT"

This mapping stores events that appear as annotations in the PVTM GUI's time series canvas. The mapping has three main properties:

- "time" : a long property with an epoch timestamp (number of seconds since Jan 1st 1970)
- "shortText": a one or two-character code displayed in the annotations in the PVTM GUI's time series graphs.
- "longText": a longer string that appears when users hover the mouse over the annotations in the PVTM GUI's time series graphs.

"CONFIG"

This mapping stores configuration changes pushed from the PVTM GUI to PVTM Agents. This mapping has three main properties:

- "time" : a long property with an epoch timestamp (number of seconds since Jan 1st 1970)
- "description": a short description of what was changed in the configuration.
- "configStr": a BASE 64 encoded string with all the contents of the PVTM Configuration file.

PVTM Elastic Search Queries

By default, the elastic search engine will listen to port 9200 for queries. Queries can be made from web browsers, or from the command line on Linux or Windows (with Cygwin or a Linux-like shell) by using the curl command.

The format of the queries is typically the following:

```
http://<PVTMSimHost>:9200/<PVTMAgentName>/<Queries>
```

The following sections show useful queries using localhost as the PVTM Simulation Host and using leo2-pc as the PVTM Agent name:

Dumping the current schema:

The following query dumps the current schema for the leo2-pc index:

```
$ curl -XGET http://localhost:9200/leo2-pc
```

Here is a sample output:

```
% Total      % Received % Xferd  Average Speed   Time    Time     Time  Current
                               Dload  Upload   Total   Spent    Left   Speed

100 2763 100 2763    0     0  179k      0  --:--:--  --:--:--  --:--:--  179k

{"leo2-
pc":{"aliases":{},"mappings":{"RUN_SOLUTION_CHECK_CACHED":{"properties":{"normReqHash":{"type":"string"},"normResults":{"properties":{"coreIds":{"type":"long"},"numaZoneIds":{"type":"long"},"pid":{"type":"long"},"threadIds":{"type":"long"}}},"score":{"type":"long"},"time":{"type":"long"}}},"RUN_SOLUTION_CHECK_CACHED_REQUEST":{"properties":{"normReq":{"properties":{"computerJson":{"properties":{"properties":{"contextSwitchCost":{"type":"string"},"id":{"type":"string"},"junkCores":{"type":"string"},"latencyMatrix":{"type":"string"},"physicalCpuList":{"type":"string"},"type":{"type":"string"}}},"type":{"type":"string"}}},"normAppLinksJson":{"properties":{"sTid":{"type":
```

```
"string"},"sourceId":{"type":"string"},"tTid":{"type":"string"},"targetId":{"type":"string"},"type":{"type":"string"}}},"normAppsJson":{"properties":{"properties":{"properties":{"allowedCores":{"type":"string"},"assumeAllThreadsTalk":{"type":"string"},"id":{"type":"string"},"pvtmCurrentCores":{"type":"string"},"pvtmThreadIds":{"type":"string"},"threads":{"type":"string"}}},"type":{"type":"string"}}}}},"normReqHash":{"type":"string"},"reason":{"type":"string"},"score":{"type":"long"},"time":{"type":"long"}}},"SET_COMPUTER":{"properties":{"computer":{"properties":{"properties":{"contextSwitchCost":{"type":"string"},"id":{"type":"string"},"junkCores":{"type":"string"},"latencyMatrix":{"type":"string"},"physicalCpuList":{"type":"string"}}},"type":{"type":"string"}}},"time":{"type":"long"}}},"SET_APPLINKS":{"properties":{"appLinks":{"properties":{"properties":{"properties":{"allowedCores":{"type":"string"},"assumeAllThreadsTalk":{"type":"string"},"id":{"type":"string"},"pvtmCurrentCores":{"type":"string"},"pvtmThrdNames":{"type":"string"},"pvtmThreadIds":{"type":"string"},"threads":{"type":"string"}}},"sTid":{"type":"string"},"sourceId":{"type":"string"},"tTid":{"type":"string"},"targetId":{"type":"string"},"type":{"type":"string"}}},"score":{"type":"long"},"time":{"type":"long"}}},"RUN_SOLUTION_STATUS":{"properties":{"normReqHash":{"type":"string"},"score":{"type":"long"},"time":{"type":"long"}}},"SET_APPS":{"properties":{"CPU":{"type":"long"},"apps":{"properties":{"properties":{"properties":{"allowedCores":{"type":"string"},"assumeAllThreadsTalk":{"type":"string"},"id":{"type":"string"},"pvtmCurrentCores":{"type":"string"},"pvtmThrdNames":{"type":"string"},"pvtmThreadIds":{"type":"string"},"threads":{"type":"string"}}},"type":{"type":"string"}}},"score":{"type":"long"},"time":{"type":"long"}}},"settings":{"index":{"creation_date":"1446229293642","uuid":"_Mz6zzKVQSa rmCuv9ExZw","number_of_replicas":"1","number_of_shards":"5","version":{"created":"2000099"}}},"warmers":{}}}}
```

Find the Index and Node Shards

Elastic search splits the data amongst shards that are hosted by a number of different instances. Occasionally, if an instance is no longer used, it is required to re-organize which indices belong to which shards, and to move them to the current one. This can be done with two operations:

- 1) Find the current shards:

```
curl -XGET http://localhost:9200/_cat/shards
```

- 2) Re-home the shards: - we provide a script to re-home shards under `<install_dir>/4.7.0/linux/es-move-shards.sh`

This script takes two arguments:

```
./es-move-shards.sh <index to move> <node to move to>
```

The output of step (1) will be similar to this:

```
curl -XGET http://localhost:9200/_cat/shards
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left  Speed
100  4020  100  4020    0     0  10720      0 --:--:-- --:--:-- --:--:-- 10720
localhost.localdomain 2 p STARTED      3648 668.6kb 127.0.0.1 Blur
localhost.localdomain 2 r UNASSIGNED
localhost.localdomain 3 p STARTED      3629 629.5kb 127.0.0.1 Blur
localhost.localdomain 3 r UNASSIGNED
localhost.localdomain 4 p STARTED      3673 645.9kb 127.0.0.1 Blur
localhost.localdomain 4 r UNASSIGNED
```



```
localhost.localdomain 1 p STARTED      3571 631.6kb 127.0.0.1 Blur
localhost.localdomain 1 r UNASSIGNED
localhost.localdomain 0 p STARTED      3635 624.5kb 127.0.0.1 Blur
localhost.localdomain 0 r UNASSIGNED
```

From this, the current node name for 127.0.0.1 is Blur. To move all the data from an index that was previously active in an elastic search node that is down to Blur, you could simply run the following command:

```
./es-move-shards.sh <agent name> Blur
```

Find all the simulation results between two Timestamps

Sometimes it is useful to query data directly from elastic search; the following query shows all the RUN_SOLUTION_CHECK_CACHED results between two timestamps (epoch in milliseconds) from an agent called 'leo2-pc':

```
curl -XPOST "http://localhost:9200/leo2-pc/RUN_SOLUTION_CHECK_CACHED/_search" -d'
{
  "query" : {
    "range" : {
      "time" : {
        "from" : 1455066061526,
        "to" : 1455152461526,
        "include_lower" : true,
        "include_upper" : true
      }
    }
  }
}
```

Here are the normalized results, which show the "normResults" value for two processes with pid 0 and pid 1:

```
% Total      % Received % Xferd  Average Speed   Time    Time       Time  Current
             Dload  Upload  Total      Spent    Left     Speed
0           0         0         0         0         0         0         0         0
"took" : 8,
"timed_out" : false,
"_shards" : {
  "total" : 5,
  "successful" : 5,
  "failed" : 0
},
"hits" : {
  "total" : 1,
  "max_score" : 1.0,
  "hits" : [ {
    "_index" : "leo2-pc",
    "_type" : "RUN_SOLUTION_CHECK_CACHED",
    "_id" : "AVLkoEKd-GYkw2WINmfY",
    "_score" : 1.0,
    "_source":{ "time":
1455098446493,"normReqHash":"ac926dba97cbfa8c464fbc1a6f670054b3eb9a4191e89d5fc6f981
2c8ec3a44c","score":"10408","normResults":[{"pid":0,"threadIds":[0,1,2,3,4,5,6,7,8,
9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,
37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64
,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91],
"coreIds":[3,3,3,1,3,2,1,2,1,1,1,3,1,0,0,0,1,0,1,2,0,0,2,2,0,0,0,3,1,2,1,3,0,3,3,3,
```


- `-Xmx2048m`
This controls the maximum memory size of the Java Virtual Machine; this value should be approximately 500MB-1GB per agent being monitored.
- `-javaagent:/home/leo/pontus-vision/4.7.0/linux/pvtm-agent.jar`
This optional command line argument
- `-Dpv.threadmgr.configFile=""`
This option allows an optional configuration file to be passed to PVTM Simulator to control the simulator's behaviour; this option should only be used after engaging with Pontus's professional services team.
- `-Dpv.threadmgr.httpPort=8444`
This option selects the port number that the agents use to communicate with PVTM Thread Manager.
- `-Dpv.elasticsearch.node.config.url=file://<path to the pvtm.yml file>`
This option selects an overriding path to configure the embedded elastic search engine
- `-Dpvtm.websockets.maxIdleTimeout= 3000000`
This option controls the maximum amount of idle time on a web socket (in milliseconds)
- `-Dpv.threadmgr.num.term.early.process.threads=0`
This option controls the number of threads used to post-process simulation results that were terminated early.
- `-Dpv.threadmgr.return.terminated.early.results=true`
This option controls whether simulation results that were terminated early should be sent back to PVTM Agent. Simulations usually get terminated early if PVTM Agent sends a STOP_SOLUTION request message whilst a simulation is currently being run. This typically happens in environments where there is a lot of volatility in the PVTM Scores.
- `-Dpv.threadmgr.jmsDynamicCommands=true`
This option selects whether or not PVTM Simulator will use a JMS broker to send/receive commands from PVTM GUI destined to PVTM Agent.
- `-Dpontus.agent.broker.name="tcp://localhost:61616"`
This option controls the JMS Broker name used by PVTM Simulator to communicate with PVTM GUI and send/receive dynamic commands to PVTM Agent.
- `-Dpv.threadmgr.to.gui.topic="pvtm.threadmgr.to.gui.topic"`
This option controls the JMS Topic used to send data from PVTM Simulator to PVTM GUI.
- `-Dpv.threadmgr.to.gui.topic="pvtm.threadmgr.from.gui.topic"`
This option controls the JMS Topic used to send data from PVTM GUI to PVTM Simulator.
- `-Dpontusvision.license.info="PontusVision.*"`
This option selects the license information for PVTM Simulator. This value is customer-specific, and is typically entered as part of the installation process.
- `-Dpontusvision.license.holder="o=temp,ou=license,cn=test123"`
This option selects the license key information for PVTM Simulator. This value is customer-specific, and is typically entered as part of the installation process.
- `-Dpontusvision.license.file="~/pontus-vision/4.7.0/license.lic"`
This option selects the license file information for PVTM Simulator. This value is customer-specific, and is typically entered as part of the installation process.
- `-Dpvtm.websockets.maxTextMessageBufferSize=8388608`
This option selects the maximum web sockets message size to be sent and received by PVTM Simulator. This value only needs to be increased in environments where the agent is running on servers with high number of threads (1000 +), or high number of inter-thread communication connections (10000+)
- `-Dpvtm_history_prefix="/home/leo/pontus-vision/4.5.0/pvtm-storage/"`
This option selects the folder where the embedded Elastic Search engine will store its data. This should be a local disk with as much storage as possible.

- `-Dpvtm.normalized.solution.level3Cache="true"`
This option determines whether or not PVTM Simulator searches for previously stored normalized solutions across different PVTM Agent indices.
- `-Dpvtm.normalized.solution.level2Cache="true"`
This option determines whether or not PVTM Simulator searches for previously stored normalized solutions for the same PVTM Agent index.
- `pv.elasticsearch.node.create="true"`
This option determines whether or not PVTM Simulator creates new elastic search nodes.
- `-cp "/home/leo/pontus-vision/4.7.0/vision-server.jar"`
This selects the path to the vision-server.jar file (this is typically filled out by the installer).
- `com.pontusnetworks.threadmgt.app.ThreadMgrWebSocketsSvc`
This is the class name that starts the web sockets version of PVTM Simulator.

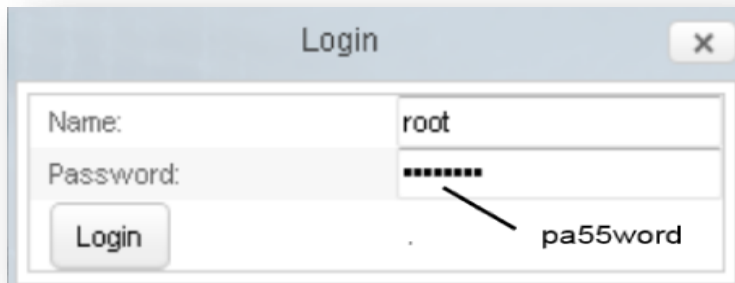
Chapter 7. PVTM GUI

PVTM GUI is a pure Java application that typically runs on the same server as PVTM Simulator. PVTM GUI allows users to query the data stored in PVTM Simulator, to run off-line simulations, and also control aspects of PVTM Agent. This chapter describes how this functionality works, as well as administrative tasks, such as logging in, and changing passwords. PVTM GUI currently uses the Pontus Vision framework, which is a full configuration management system. For brevity, the scope of this document will focus mostly on the PVTM-related functionality. The following sections describe how each of the PVTM-related parts of the GUI:

Login Screen

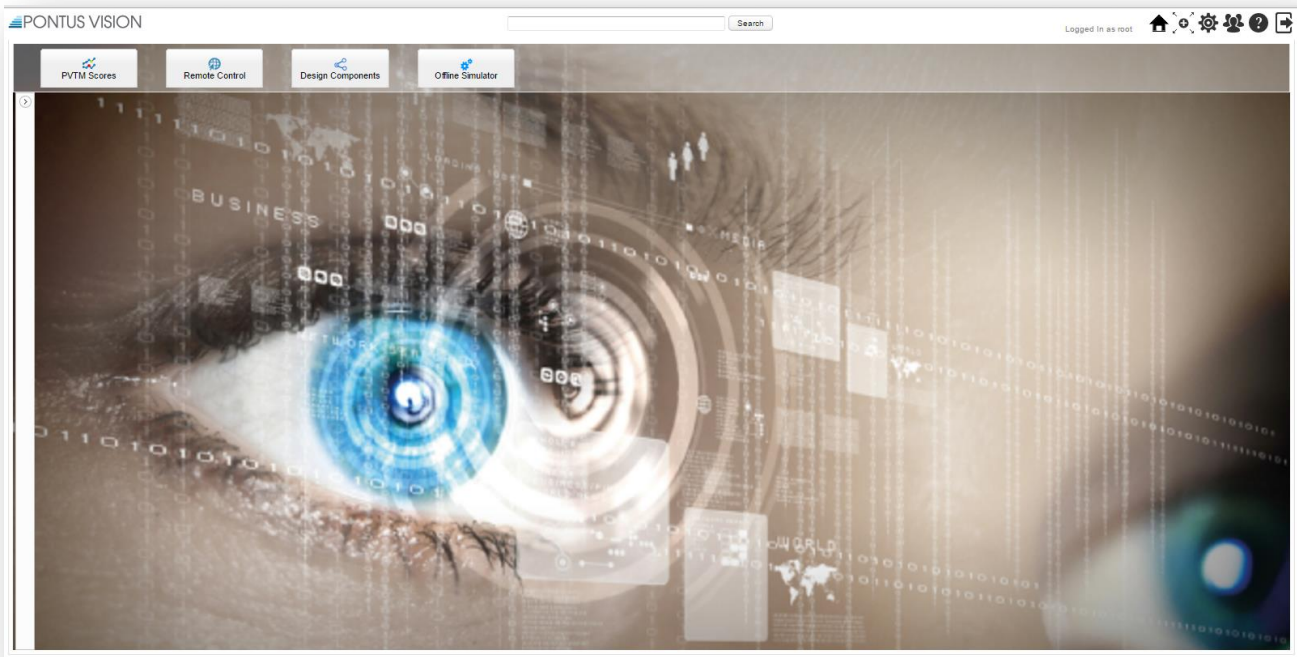
As soon as PVTM GUI is started (see Starting PVTM Simulator and GUI), users can login to the system using the appropriate URL from a Browser (the default URL is <https://localhost:8443/>). If logging in for the first time, the following user name and password should be used:

User: root
Pass: pa55word



As soon as the login is successful, the next screen that appears is the Main Canvas.

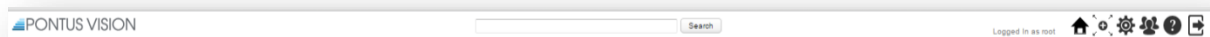
Main Canvas









The Main Canvas allows users to navigate throughout the user interface. By default, as soon as a user logs in, an instance of The Time series Graph Window appears, along with a splash screen showing the latest release notes. Closing both of these windows reveals the Main Canvas.

The Main Canvas has two main areas: the Top Toolbar, and the Navigation Buttons.

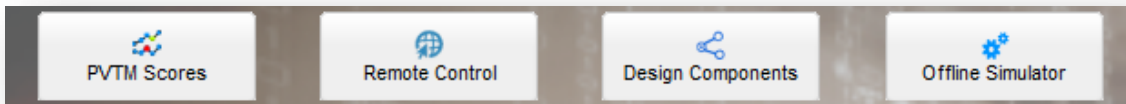
Top Toolbar







The Top toolbar allows users to explore a number of Pontus Vision framework features that are outside the scope of this document. These will only be briefly mentioned here for completeness:

- The Search Button located in the top centre of the top toolbar enables users to search the Pontus Vision Framework's templates, knowledge base, and configuration placeholders.
- The Home Icon  takes users back to the main canvas PVTM screen
- The Min/Max Icon  takes users to the PontusVision Framework view, which is out of scope for this document. To return to the main canvas, either click on the Home Icon.
- The Settings Icon  enables users to modify PontusVision Framework settings, which are currently out of scope for this document.
- The Users Icon  allows users to change the current user's password; it brings up the window in the Changing the Password section.
- The Information Icon  brings up the current license information, and also allows users to update the license key without restarting PVTM (see Updating the license) for more details.
- The Logout Icon  takes users back to the Login Screen.

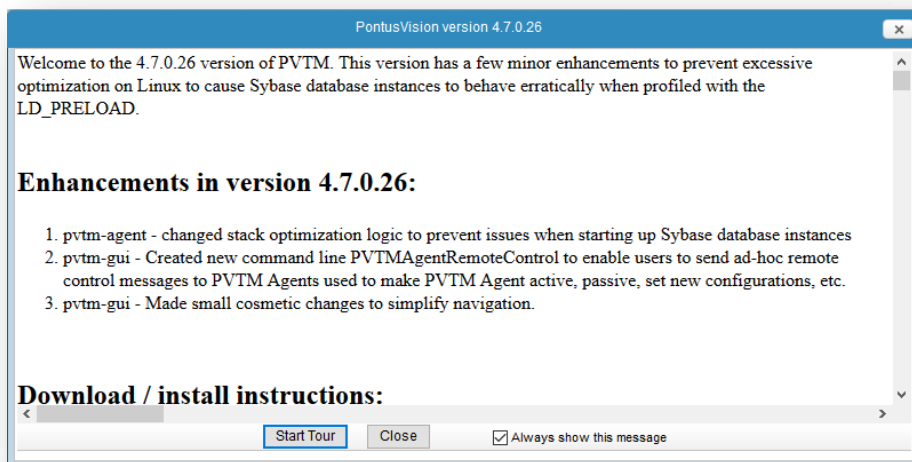
Navigation Buttons



Four main buttons allow users to Navigate to the main thread manager windows:

- The PVTM Scores  button brings up new instances of The Time series Graph Window.
- The Remote Control  button brings up new instances of The Remote Control Window
- The Design Components  button brings up new instances of The Design Components Window
- The Offline Simulator Button  brings up new instances of The Offline Simulator Window, in offline simulation mode. This enables users to build their own applications using custom software and hardware components without having previously run any PVTM Agents

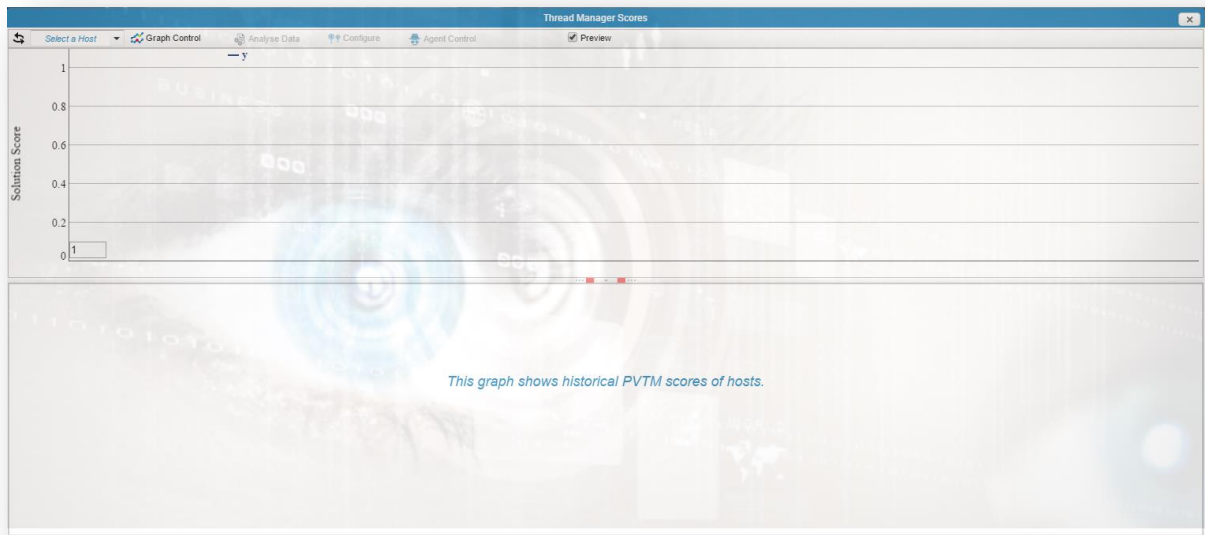
The Release notes Splash Screen



The release notes splash screen shows details of bug fixes and enhancements in the latest release, as well as previous releases. This window has three basic buttons:

- Start Tour – starts a step-by-step guide on how to use The Time series Graph Window
- Close – closes the window
- Always show this message – determines whether this screen will appear in the future for this user.

The Time series Graph Window



The Time series Graph Window shows historical scores for servers where PVTM Agent is running. This window automatically appears as soon as users login to the PVTM GUI. Each window shows the historical and semi-real-time scores for a single host. To view scores from multiple servers, several instances of this window may be opened by clicking on The Thread Manager Scores button in the Navigation multiple times.






This window has three main areas: The Time series Toolbar, The Time series Canvas, and The Preview Area.

The Time series Toolbar



The time series toolbar has several widgets:

- Refresh ↻ Button – refreshes The Time Series Canvas with the latest data, and also refreshes the list of available hosts, as well as the Agent Control status
- **Select a Host** combo box – enables users to select a time series for any hosts running a PVTM Agent. Here are some notes about this widget:
 - The drop-down will be empty until at least one PVTM Agent runs.
 - If selecting a host does not produce a graph, the PVTM Agent on that host may be inactive, and data may have been available in the past; try selecting a larger Historical Time Window.
 - Starting in version 4.7.0.13, hovering the mouse on top of this widget opens it up; it is no longer necessary to click on it.
- Graph Control 🎨 Menu Button– As soon as users click on this menu button, the Graph Types popup window opens with the following check boxes:

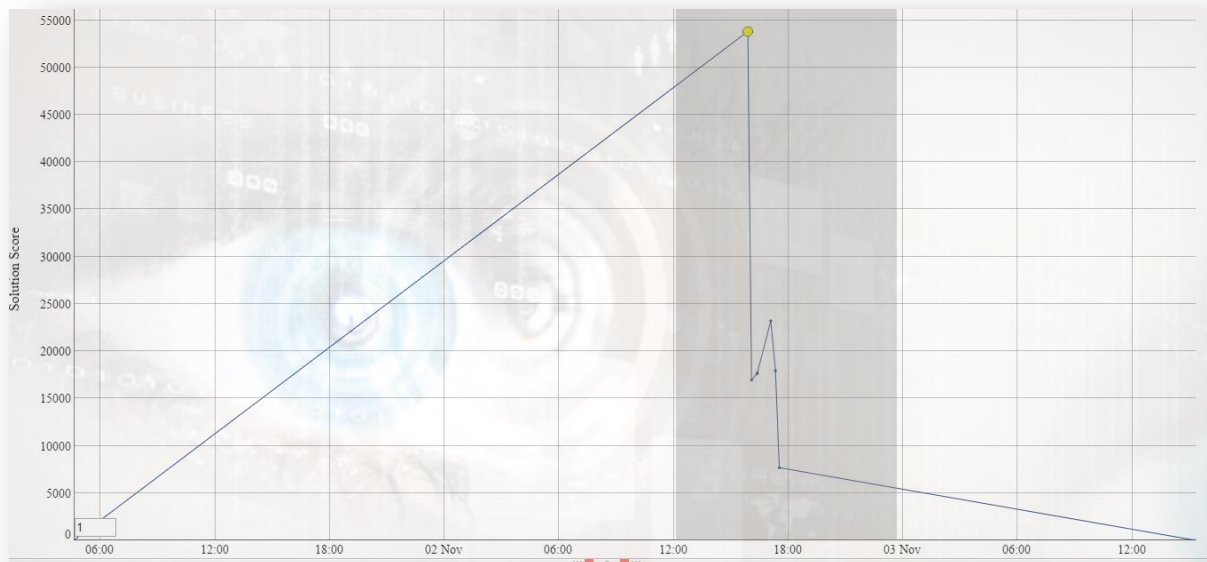
- Total CPU % checkbox – enables users to overlay the total CPU % utilization across all threads being monitored on top of the score time series. Note that the legend for the CPU % utilization appears on the right-hand side of The Time Series Canvas
- Thread Pinning Points checkbox– overlays green dots on The Time series Canvas that represent a point in time when a thread pinning request was made by the agent, and overlays red dots that represent a point in time when a thread pinning solution was applied by the agent.
- Auto Refresh checkbox – polls the time series data every 5 seconds, updating the Date box to the current time, which in turn refreshes The Time series Canvas.
- Date box – enables users to specify a point in time for the right-most data item in the graph. The Historical Time Window selects the left-most point.
- Historical Time Window (5 minutes) – enables users to choose different time spans to view the time series data starting from the Start Time.
- Zoom Out  Button – resets the zoom in the Time Series Canvas to match the currently selected Start Time and Historical Window boundaries.
- Log Scale checkbox - changes the y-axis of the time series into a log scale when checked, and a linear scale when unchecked. This is helpful to compare many graphs that are orders of magnitude apart.
- Show annotations checkbox - when checked, it adds annotations to important events in the time series. The annotations have one or two characters; here are a few examples:
 - **B** - begin of a new session - this appears whenever PVTM Agent connects to the PVTM Simulator. Note that when a configuration is changed, this can appear multiple times as the agent connects and disconnects to apply the new configuration.
 - **E** - end of a session - indicates that PVTM Agent has disconnected from the PVTM Simulator.
 - **C** - new configuration - indicates that a new configuration has been applied.
 - **Ru** - run solution - indicates that a new set of run solution parameters has been applied (see `pvtm_agent_run_solution_command_after_new_process_delay` for more details of when this can happen).
 - **M** - multiple - indicates that multiple events have occurred at a particular point in time. This typically appears when the Analyse Data button has been pressed.
 - **oc** - overloaded core - indicates that a core has more than one active thread
 - **tc** - thread created - indicates that a thread has been created.
 - **td** - thread destroyed - indicates that a thread has been destroyed.
- Configure  Button - changes the Preview Area into configuration mode. This enables users to review the existing configuration, modify it, and send it out to an individual PVTM Agent.
- Analyse Data  Button - takes the existing time series displayed, and analyses all the historical data for that period, highlighting events such as oversubscribed cores (e.g. cores with more than one active thread), and threads being created and destroyed. When users click this button, the bottom half of the screen shows these statistics. To revert to the normal Thread Management Preview Area, simply click on any point in the time series.
- Agent Control  Menu Button – This toolbar button serves two purposes: first, it shows the status of the agents (after the refresh button  is pressed); second, clicking on this

shows a popup window that enables users to make PVTM Agent active or passive. When PVTM Agent is active, it will automatically apply thread pinning strategies; when it is passive, it will simply collect data.

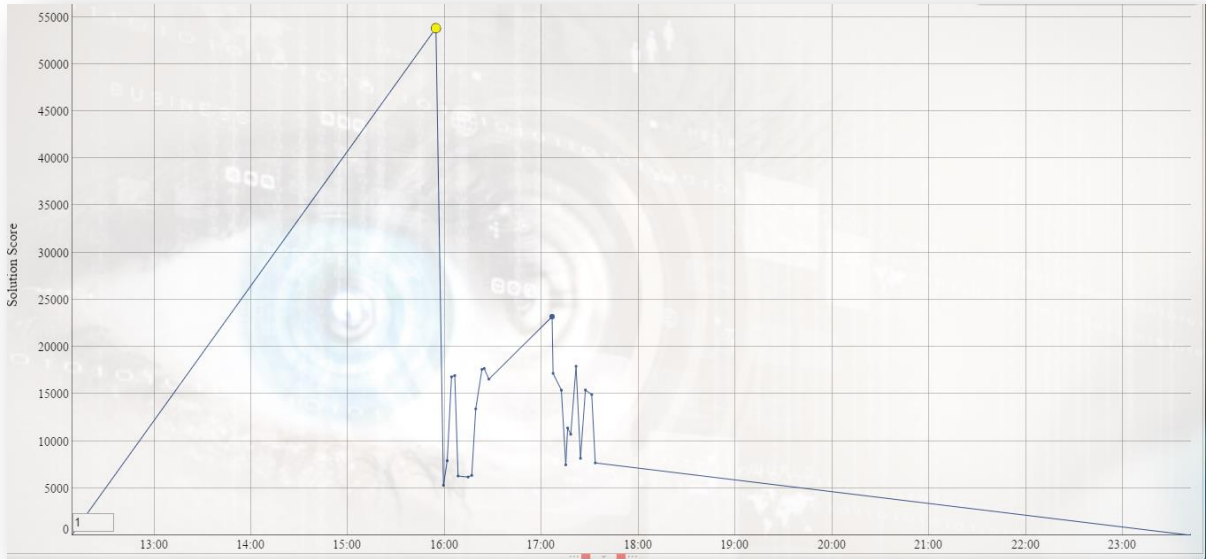
- Preview Checkbox – when checked, it opens up the panel with The Preview Area. When unchecked, it opens The Preview Area.


The Time Series Canvas

The time series canvas shows different time series depending on which toolbar widgets are selected. The time series data is summarized by taking the biggest peak values across a time period. As such, data over large time windows lose granularity; however, users can still look at more granular data by zooming in the graph. To zoom in the graph horizontally or vertically, users can drag the mouse so a grey shade appears as shown below:

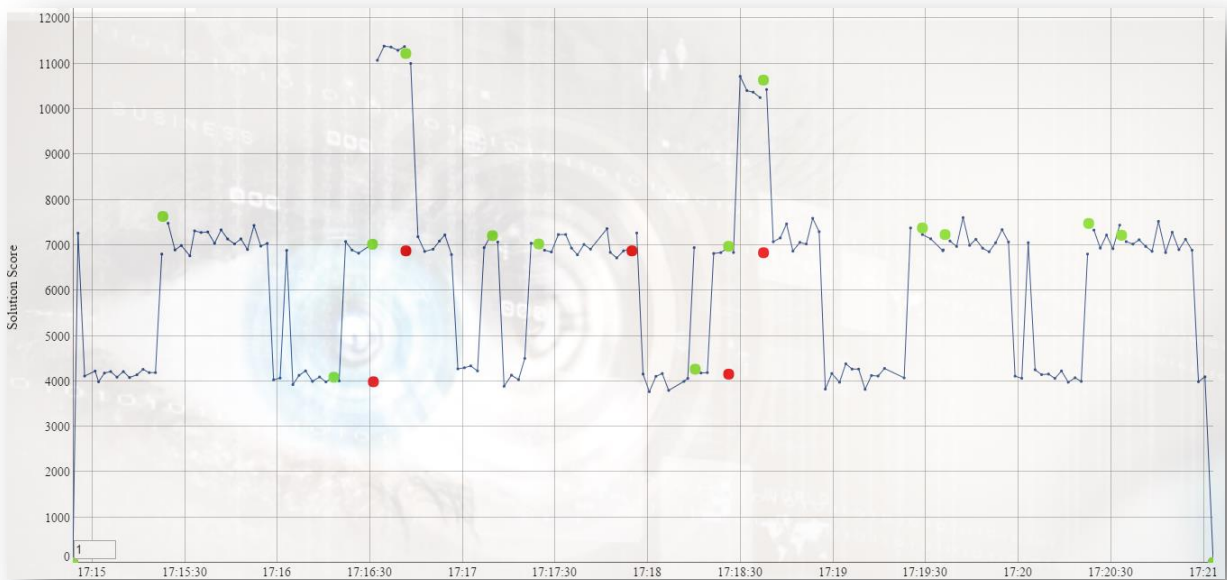


After releasing the mouse and waiting a few seconds, the example above would appear as follows:

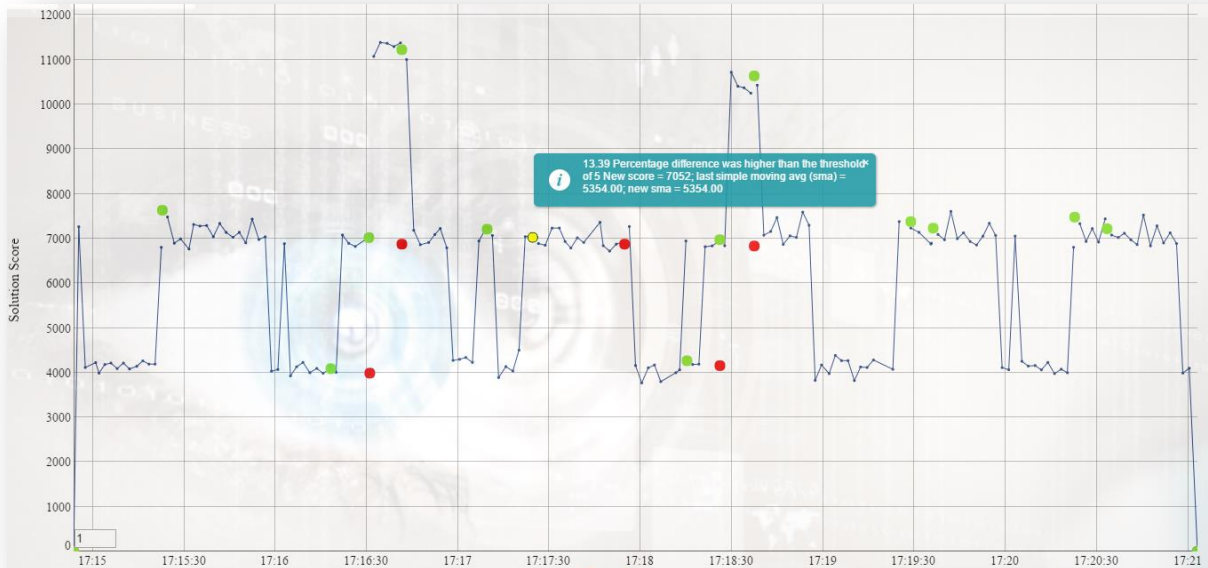


NOTE: To zoom out, double click on the canvas, or click on the Zoom Out  button.

As mentioned in The Time series Toolbar, clicking on the Thread Pinning Points checkbox causes green and red dots to appear overlaid on the screen as follows:

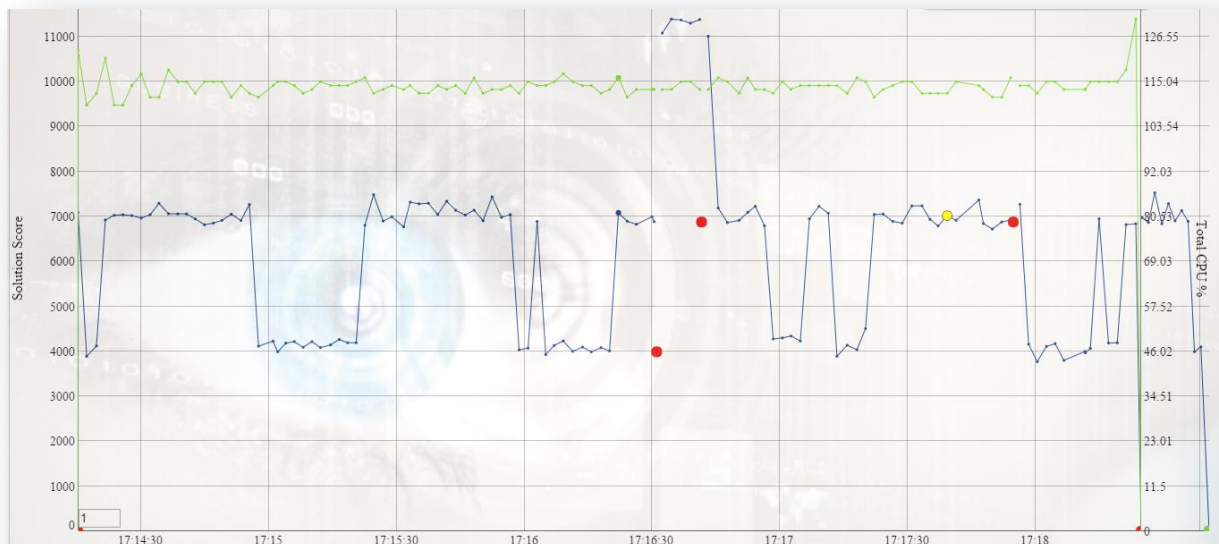


Clicking on a green dot causes the reason for the thread pinning request to appear:

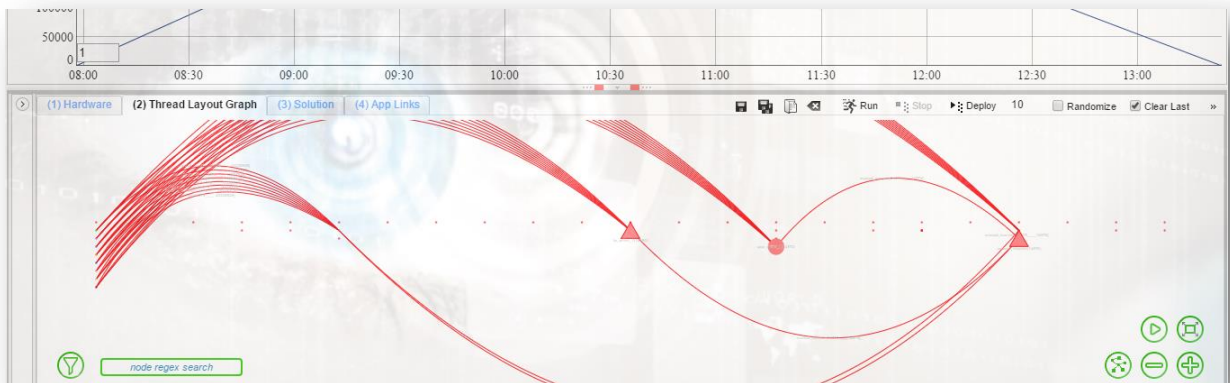


Clicking on a red dot causes the normalized thread pinning strategy that was applied at that point in time to appear inside a The Offline Simulator Window

Selecting on Total CPU % causes the green time series with total CPU Util across all monitored threads to appear (note that this causes the green dots to disappear):




The Preview Area



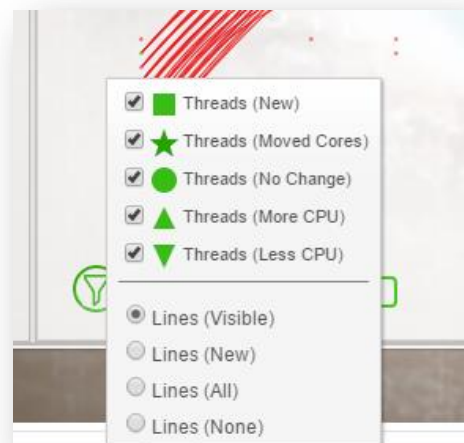
The preview area is located right below The Time Series Canvas. It enables users to see changes in status between two points in The Time Series Canvas. The Preview Area is almost identical in functionality to The Offline Simulator Window, with the key difference being the Software Tab behaviour. To avoid repetition, we will focus only on the Software Tab behaviour here, and cover the functionality of the other tabs in The Offline Simulator Window.

The first time a user clicks on a point in The Time Series Canvas, all the threads will appear as New Threads, with a square icon. As soon as a different point is clicked, the GUI will calculate the deltas from the previous point, and display the icons using the following convention:

- Squares – new threads that did not exist in the previous score
- Circles – threads that had 0 changes from the previous score
- ★ Stars – threads that moved from one core to another on the previous score
- ▲ Triangles Pointing up – threads that stayed in the same core, but **increased** CPU utilization from the previous score
- ▼ Triangles Pointing down - threads that stayed in the same core, but **decreased** CPU utilization from the previous score

Users can filter out which icons to see by Clicking on the Filter Button  in the bottom left. When clicked, the following pop-up window in the left opens. This allows users to filter not only the icon shapes, but also the links between threads:









- Lines (Visible) – this shows only the lines for the threads that are visible after the shape filters are applied.
- Lines (New) – this shows only the new lines that appeared since the last score was clicked.
- Lines (All) – this shows all the lines, and adds any missing threads filtered out by the filters above.
- Lines (None) – this suppresses all the lines.



Next to the filter button, the "node regex search" text box allows users to search for any of the text that appears as a tooltip text when hovering on top

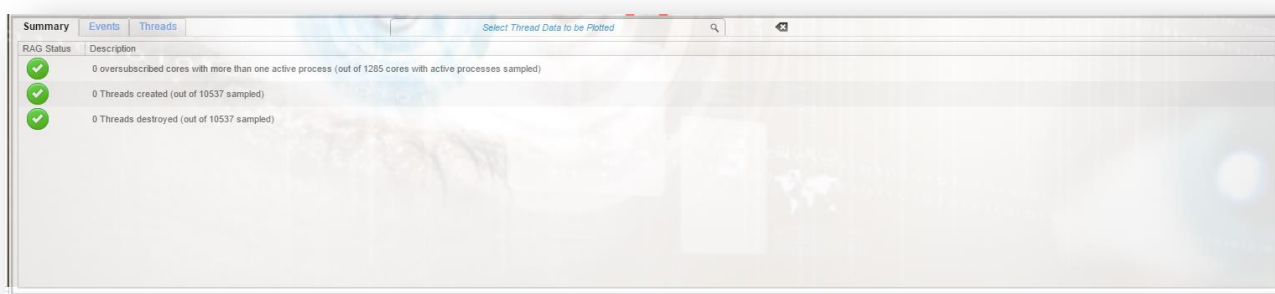
of a thread (e.g. thread names, core numbers, CPU utilization), or for specific thread shapes. To search for a specific set of shapes, users should type 'shape:' followed by a regex that matches one or more particular shapes (e.g. shape:square|star will zoom into view either stars or squares). Whenever the user hits <Enter>, the search is repeated, and the next match is zoomed into view and centred.




The five buttons on the lower right-corner control the zoom and layout of the canvas:

- The layout button  next to the minus button has 3 different states that move the threads into different layout modes; note that the transition between the Vertical and Horizontal layout goes through the cosmos layout (e.g. if the user clicks on the Vertical Core layout button, it turns into the Cosmos button; clicking then again on the Cosmos button changes it to the horizontal layout button):
 - Cosmos layout  - scatters all the threads using physics without observing the thread core position
 - Vertical Core layout  - aligns all the threads in **columns** grouped by the current thread core location (e.g. threads in core 0 go in the first column on the left, core 1 in the next column, etc.).
 - Horizontal Core layout  - aligns all the threads in **rows** grouped by the current thread core location (e.g. threads in core 0 go in the first row on the top, core 1 in the next row, etc.).
- The Plus  and Minus  Buttons zoom the canvas in and out
- The Play  button starts an animation; once clicked, it becomes the Pause button. The Pause  button stops animations, and once clicked becomes the Play button again.

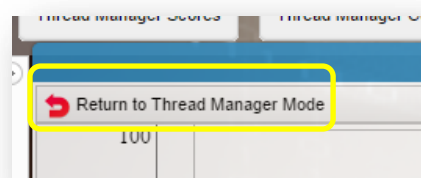


The Data Analysis Preview Area



When users click on the Analyse Data  Button, the PVTM GUI will analyse the data for each of the points shown in the time series, and find events that may impact application performance. Note that only the data points displayed are analysed, so if using a large time period, only the peak values are displayed. To get more details, simply zoom into the time series whilst in Thread Manager mode, and click on the Analyse Data  Button again. When users click the Analyse Data  Button, the PVTM GUI replaces the normal Thread Manager Preview Area with the Analyse Data Preview Area.

NOTE: To return back to the Thread Manager Mode, click on



the left-most button at the top toolbar:

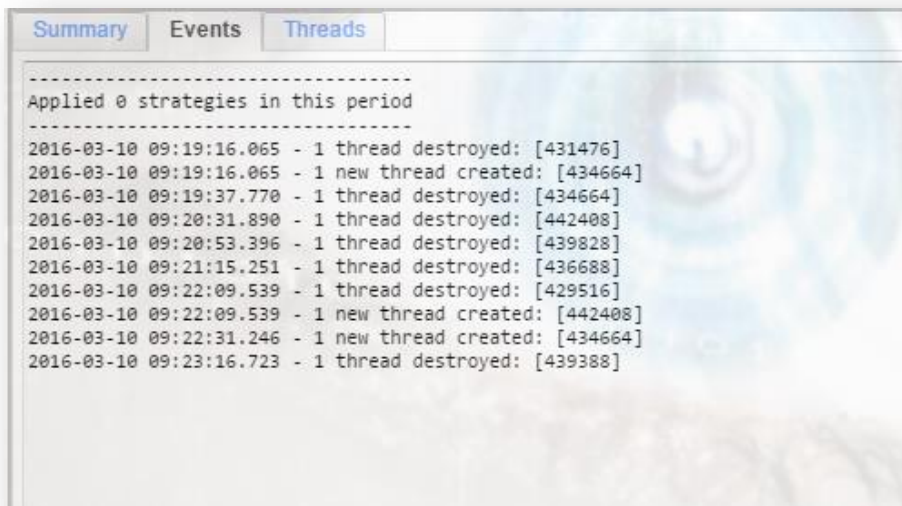
The Data Analysis Preview Area has three main tabs, and a special "Select Thread Data to be Plotted" combo box that enables users to visualize per-thread data:

Summary Tab

The Summary tab has a simple grid with two columns; the first column is a Red Amber Green (RAG) status for the environment, and the second column has a summary of the events for the current time period chosen. The following events are currently analysed:

- Core Oversubscription - this event indicates how many cores within the whole sampling period in the time series had more than one active thread running (an active thread is a thread with more than 0% CPU utilization). Cores that are oversubscribed cause performance issues in an application, and should be avoided at all costs.
- Threads Created - this event shows how many threads were created during the whole sampling period in the time series. Applications that have too many threads being created can suffer from performance issues. If at all possible, it is recommended to use fixed thread pool sizes.
- Threads Destroyed - this event shows how many threads were destroyed during the whole sampling period in the time series. Applications that have too many threads being destroyed can suffer from performance issues. If at all possible, it is recommended to use fixed thread pool sizes.

Events Tab



The Events Tab shows detailed events within the time series above, showing the timestamp when each individual event occurred.

Threads Tab

Thread ID	Description	Start Time	Run Time (ms)	Max CPU	Avg CPU	Num Oversubscribed C	Max TLB Misses	Avg TLB Misses	Max Cache Misses	Avg Cache Misses	Details
11121	QuickOrderApiSe(11121)	Mon Mar 14 10:41:55 GMT 2016	5719929	100.00	99.49	0	3526598.00	2017164.33	6651559.00	6117598.54	
11124	fx_reader(11124)	Mon Mar 14 10:41:55 GMT 2016	5719929	100.00	99.47	0	55165.00	37705.18	6538030.00	6088034.66	
11126	fx_writer_1(11126)	Mon Mar 14 10:41:55 GMT 2016	5719929	100.00	99.47	0	1012.00	112.33	51461.00	38980.85	
11123	fx_session_han(11123)	Mon Mar 14 10:41:55 GMT 2016	5719929	0.00	0.00	0	833.00	480.99	15376.00	5719.84	
11125	fx_reconnect_1(11125)	Mon Mar 14 10:41:55 GMT 2016	5719929	0.00	0.00	0	472.00	58.48	4450.00	580.24	
11122	request_respons(11122)	Mon Mar 14 10:41:55 GMT 2016	5719929	100.00	99.42	0	77.00	34.89	49254.00	35036.34	
11120	qapi_writer_1(11120)	Mon Mar 14 10:41:55 GMT 2016	5719929	100.00	99.47	0	74.00	35.33	54241.00	45967.41	
11112	java(11112)	Mon Mar 14 10:41:55 GMT 2016	5719929	0.00	0.00	0	61.00	11.70	1507.00	374.86	
11119	main(11119)	Mon Mar 14 10:41:55 GMT 2016	5719929	0.00	0.00	0	57.00	6.68	866.00	111.53	

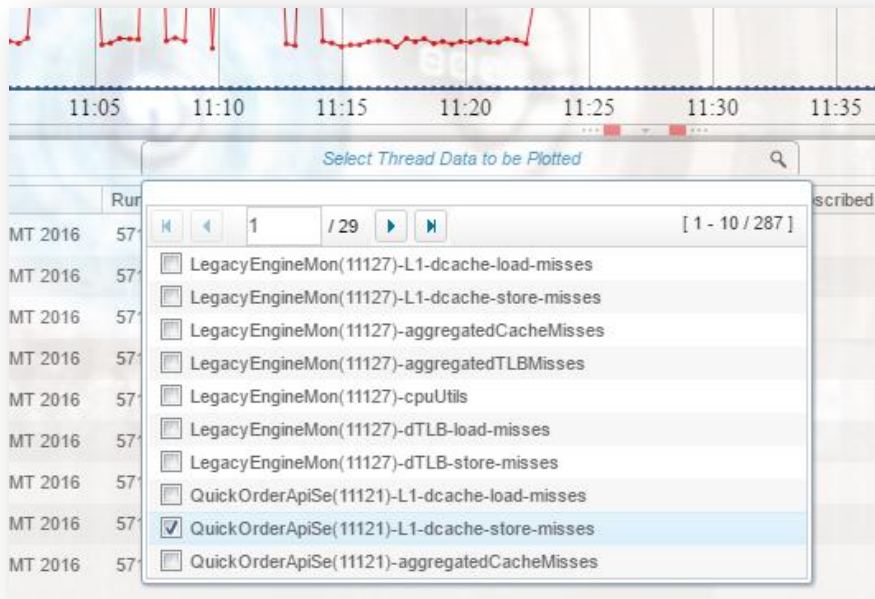
Lastly, the Threads Tab shows useful information about the threads during a period of time. The columns are sortable by clicking on the column header. The following columns are available:

- Thread ID - shows the Operating System's thread ID
- Description - in Operating Systems that support thread names, the names will be displayed. In Operating Systems that do not support this feature, the thread ID appears again in parenthesis.
- Start Time - the time within the currently shown time series when the thread first appeared.
- Run Time (ms) - the amount of time within the time series that the threads were running. This is only accurate to the time series's sampling period, which is by default 2-10 seconds.
- Max/Avg CPU - the max/average CPU utilization for the thread during the time series's time period.
- Num Oversubscribed cores - the number of times this thread was active (with more than 0% CPU utilization), and was also running in the same core as other active threads. If this number is larger than zero, it indicates an environment where thread pinning may not be very efficient unless the following option is set to a value higher than 0: `pvtm_agent_run_config_apply_pinning_per_thread_merge_last_n_solutions`
- Max/Avg TLB Misses - this shows (only on Linux-based agents) the number of times a thread was trying to read memory, and it failed to find it in the translate lookaside buffer (TLB). Note that this is an aggregated figure; the PVTM GUI will add all the performance counter time series that have the token TLB in them. Whenever a TLB miss occurs, the performance of the thread is severely affected. This can be improved by increasing the page size on the machine (e.g. on Linux, run the following commands as root:


```
vm.nr_hugepages = 10240
vm.nr_hugepages_mempolicy = 10240
```

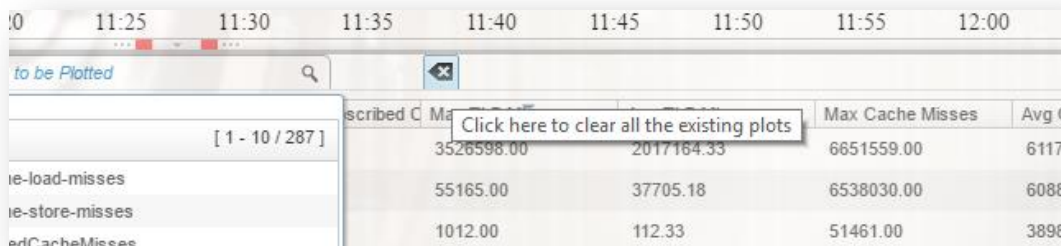
 and (if using java), setting the following JVM flags: `-XX:LargePageSizeInBytes=2m \ -XX:ReservedCodeCacheSize=320m \ -XX:+UseLargePages \`
- Max/Avg Cache Misses - this shows (only on Linux-based agents) the number of times a thread was trying to read memory, and it failed to find it in a local cache (level 1, 2 or 3). This can impact performance of applications, and can be impacted if the operating system scheduler moves a thread from one core to another. Note that this is an aggregated figure; the PVTM GUI will add all the performance counter time series that have the tokens L1 and LLC (last level cache) in them.
- Details - clicking on this button this takes the current thread, and adds it as a filter to the "Select Data To be Plotted" combo box. This then allows users to add thread-specific time series to the graph above.

Select Thread Data to be Plotted

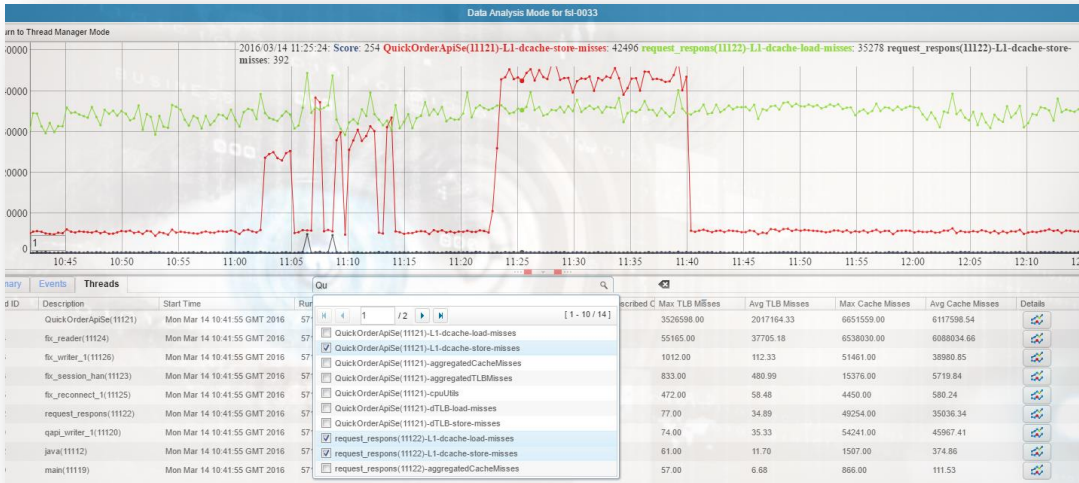


This combobox enables users to quickly search (by typing on the text field), and select various data points for all the threads currently introspected. Clicking on the checkboxes immediately adds the time series to the graph area.

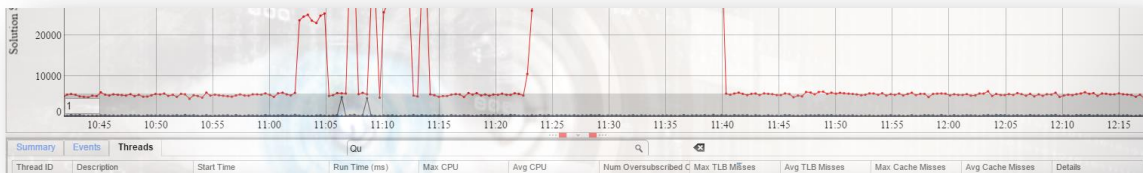
To clear individual entries, uncheck the checkbox; to clear all time series, click on the clear all button to the right of the checkbox.



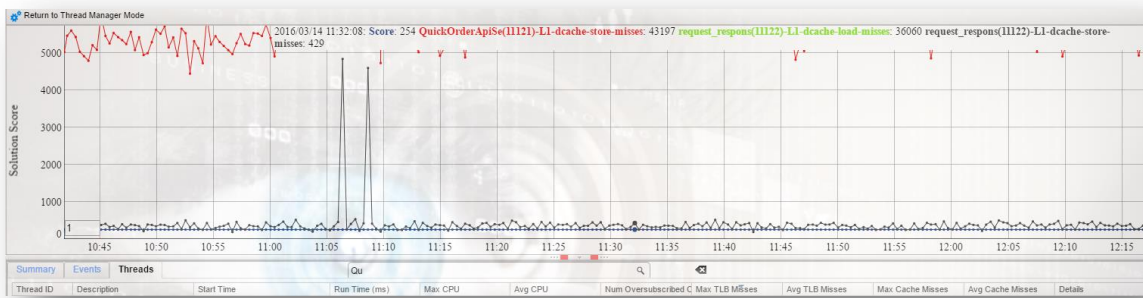
Note that the types of time series available to plot depend on the operating system, as well as on the `pvtm_agent_app_thread_perf_counter_events_csv` setting in the respective PVTM Agent's configuration file. Here's a quick example of a chart comparing different L1 cache misses for two threads:



This data is extremely useful to troubleshoot performance problems. Keep in mind that the values may have very different orders of magnitude, so users can always zoom in vertically by dragging the mouse across the canvas and unclicking to zoom into the shaded area:




Here is the zoomed-in version of the picture above:

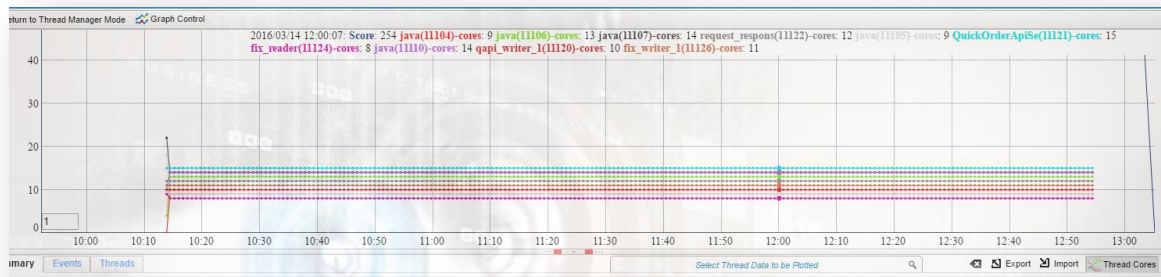


NOTE: to zoom out, double click on an empty part of the canvas.

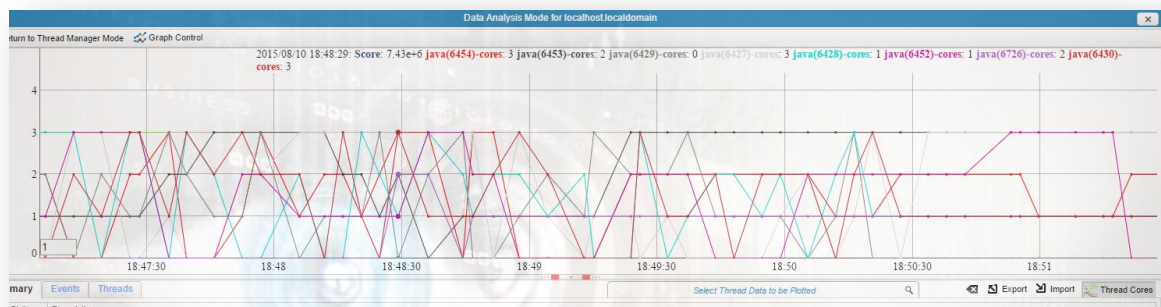
Thread Cores Button

The Thread Cores  Button takes the top 10 threads listed in the Threads Tab grid, and shows their core position in the time series. This is a toggle button; when pressed, the graph's y-axis is capped at the number of cores + 30%, and 10 time series are added to the existing time series. When the button is untoggled, the time series axis are reverted back to normal, and the time

series with the core numbers are removed. This view is invaluable to show clients how much, or how little the threads move. A well-behaved system will tend to have 'guitar string'-like lines:



In contrast, here is an example of free-flowing threads jumping between cores without thread pinning in place:



The Configuration Preview Area

The configuration Preview area has two tabs and a toolbar

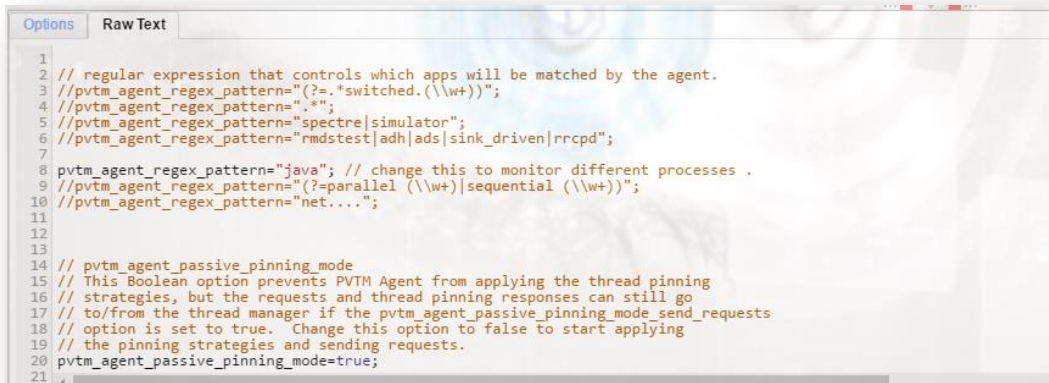
The Options Grid Tab

The screenshot shows a table with two columns: 'Name' and 'Value'. The table contains various configuration options for the PVMT agent. The options are listed in a read-only grid. The toolbar at the top includes buttons for 'Enter config description', 'Refresh', 'Import', 'Agent Deploy', and 'Agent Load'.

Name	Value
pvtm_agent_regex_pattern	java
pvtm_agent_passive_pinning_mode	true
pvtm_agent_passive_pinning_mode_send_requests	false
pvtm_agent_app_link_inet_socket_delta_threshold	0
pvtm_agent_app_link_inet_delta_threshold	0
pvtm_agent_app_link_inet_socket_delta_threshold	0
pvtm_agent_app_link_file_delta_threshold	0
pvtm_agent_app_link_inclusive_regex_filter	^(?!.*(jar dev tarandom dev random)).*\$
pvtm_agent_chrt_rr_matching_processes	true
pvtm_agent_set_chrt_rr_priority	97
pvtm_agent_ignore_zero_percentage_threads	false
pvtm_agent_inq_auto_filter_regex_pattern	ethjens

The options grid shows a read-only grid with the latest set of options displayed. Depending on when the Refresh button is pressed, new options appear in green, and values that were modified in red.

The Raw Text Editor Tab



```
Options Raw Text
1
2 // regular expression that controls which apps will be matched by the agent.
3 //pvtm_agent_regex_pattern="(?!.*switched.(\w+))";
4 //pvtm_agent_regex_pattern=".*";
5 //pvtm_agent_regex_pattern="spectre|simulator";
6 //pvtm_agent_regex_pattern="rmdstest|adh|ads|sink_driven|rrcpd";
7
8 pvtm_agent_regex_pattern="java"; // change this to monitor different processes .
9 //pvtm_agent_regex_pattern="(?!parallel (\w+)|sequential (\w+))";
10 //pvtm_agent_regex_pattern="net...";
11
12
13
14 // pvtm_agent_passive_pinning_mode
15 // This Boolean option prevents PVTM Agent from applying the thread pinning
16 // strategies, but the requests and thread pinning responses can still go
17 // to/from the thread manager if the pvtm_agent_passive_pinning_mode_send_requests
18 // option is set to true. Change this option to false to start applying
19 // the pinning strategies and sending requests.
20 pvtm_agent_passive_pinning_mode=true;
21
```

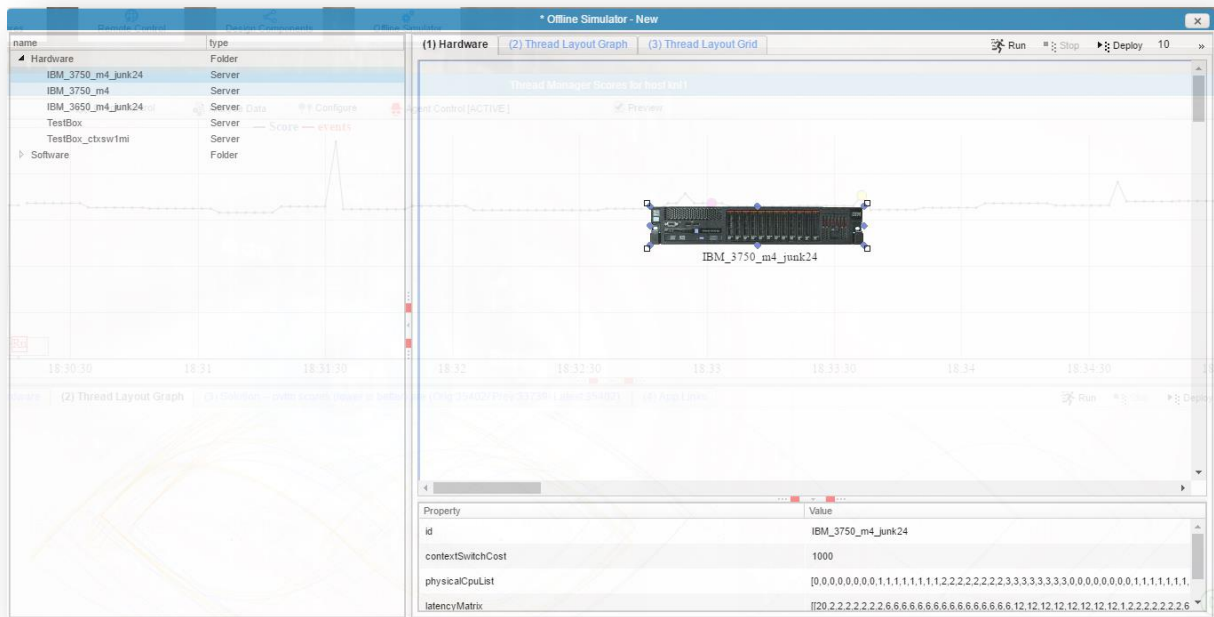
The raw text editor enables users to edit the configuration. Hitting CTRL+Space brings a pop-up combo box with all the valid option names (see pvtm.cfg Configuration File for more details).

The Configuration Preview Toolbar



The configuration Preview Toolbar enables users to import external configuration files, deploy configurations to the current PVTM Agent, or to load the current configuration in the agent. The Refresh button enables users to refresh the options grid with the latest values from the raw text editor. When users press the Refresh button, the grid will show options that did not change in black, new options in green, and options that changed in red.

The Offline Simulator Window



The Offline Simulator Window has most of the same functionality as the Preview Area; whenever the Preview checkbox is unchecked, it pops up in a separate window.

The Offline Simulator Window can perform three main functions: Monitoring, Simulation, and Deployment. If the Offline Simulator Window is started by clicking on one of the blue dots in The Time Series Canvas, it starts in Monitoring mode. If, however it is started by clicking on the Thread Manager button in the Thread Manager Navigation tab, the Offline Simulator Window starts in a pure Simulation-mode, where users can use custom-built widgets to run hypothetical what-if analysis in the system. The following sections show how each of these functions work, within each of the main Window Parts:

Offline Simulator Window Parts

The Offline Simulator Window has three main areas: the Thread Pinning Layout Toolbar, the Thread Pinning Layout Tabs, and the Hardware and Software Components Tree.





Thread Pinning Layout Toolbar

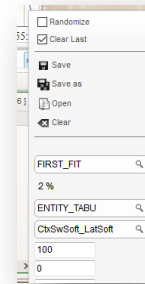


- Run – starts an off-line simulation for X seconds (default 30)
- Stop – stops a currently running simulation
- Deploy – sends the current thread pinning layout in The Solution Tab to the relevant PVTM Agent for immediate effect. This forces the simulation to be applied immediately.

Note that this allows users to even send a manually-created thread pinning strategy to PVTM Agent. The PVTM Agent is always the one associated with the current Offline Simulator Window's original Time series graph.

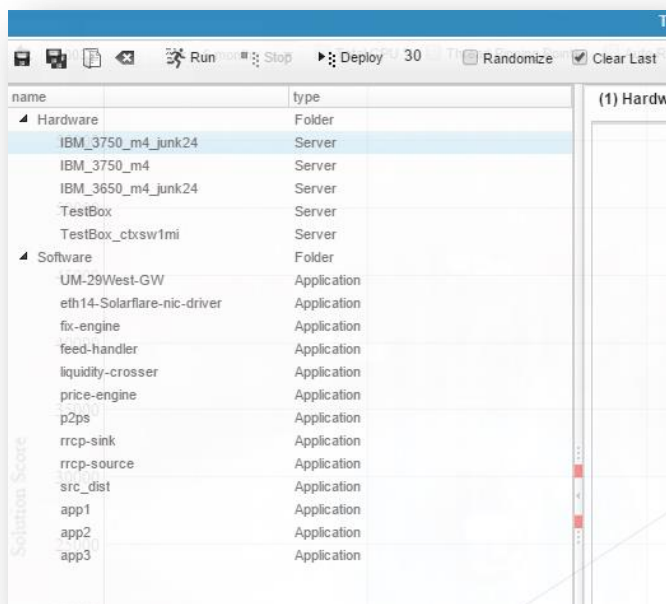
- Time textbox– determines the number of seconds to run a simulation (defaults to 10 seconds).
- Double Chevron **»** – expands the toolbar to enable users to select less common actions:

- Randomize checkbox – if clicked, randomizes the initial placement of threads in cores when running a new simulation. This is useful for larger models if threads are getting stuck in local optima.
- Clear Last checkbox – clears the last layout and uses one of the initial phase algorithms to place the threads on cores initially.
- Save  – allows users to save a copy of the current simulation results as scripts that can start applications on the right set of threads.
- Save as  – same as save, but if a new copy is required
- Open  – open a previously saved simulation
- Clear  – clear the current simulation information
- Advanced Items - allows users to control some of the simulation parameters (See Thread Execution Layout Simulation for more details)

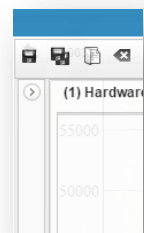


Hardware and Software Components Tree

The Hardware and Software Components Tree are located to the left of the Thread Pinning Window, and can either appear in a collapsed format (when in monitoring mode), or in expanded format (when in simulation mode).



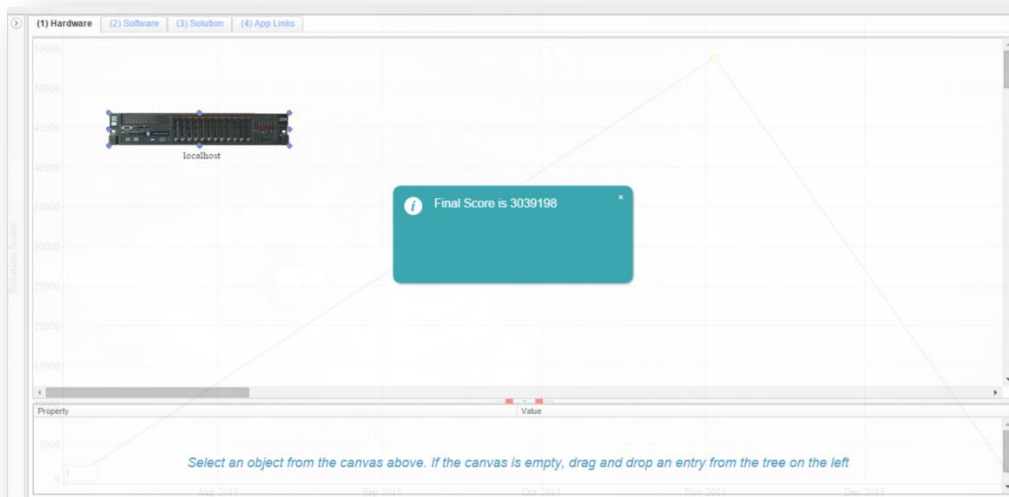
If in collapsed format, it can be expanded by clicking on the small arrow below the Thread Pinning Layout Toolbar (as seen on the figure on the right).



If in expanded format, it can be collapsed by clicking on the arrows next to the red border between the Hardware and Software Components Tree and the Thread Pinning Layout Tabs (as seen on the figure on the left). When running in expanded mode, clicking on the hardware or software parts of the tree will automatically choose the Hardware or Software tabs.

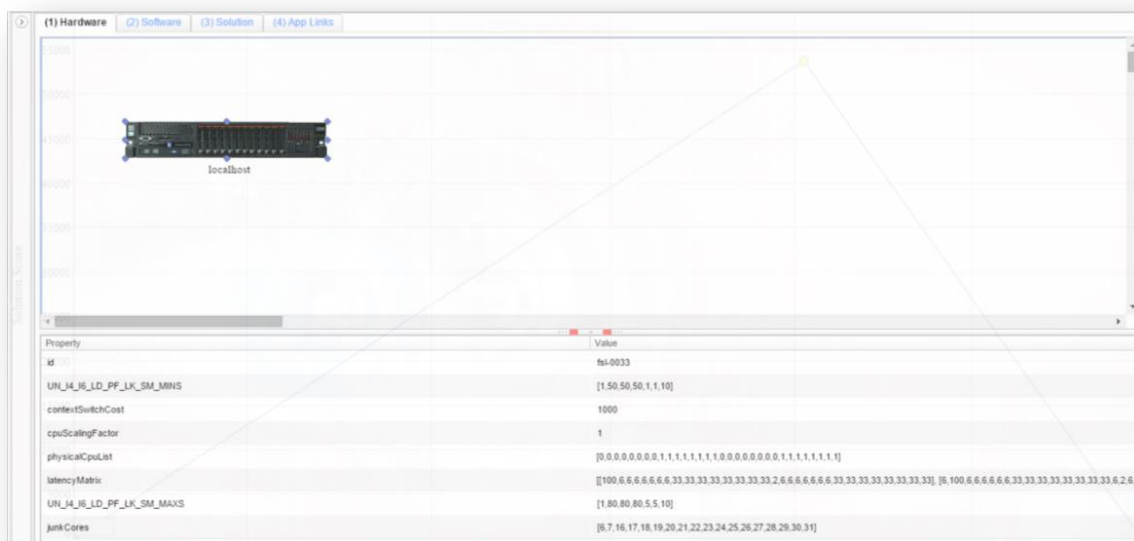
Thread Pinning Layout Tabs

The Thread Pinning Layout Tabs show different types of tabs depending on which mode is currently active.



If users click on one of the lines in The Time Series Canvas, this window starts in monitoring mode, displaying the current state of the environment at a particular point in time. When in monitoring mode, the window has 4 main tabs: The Hardware Tab, The Thread Layout Tab, The Solution Tab, and The App Links Tab.

Hardware Tab



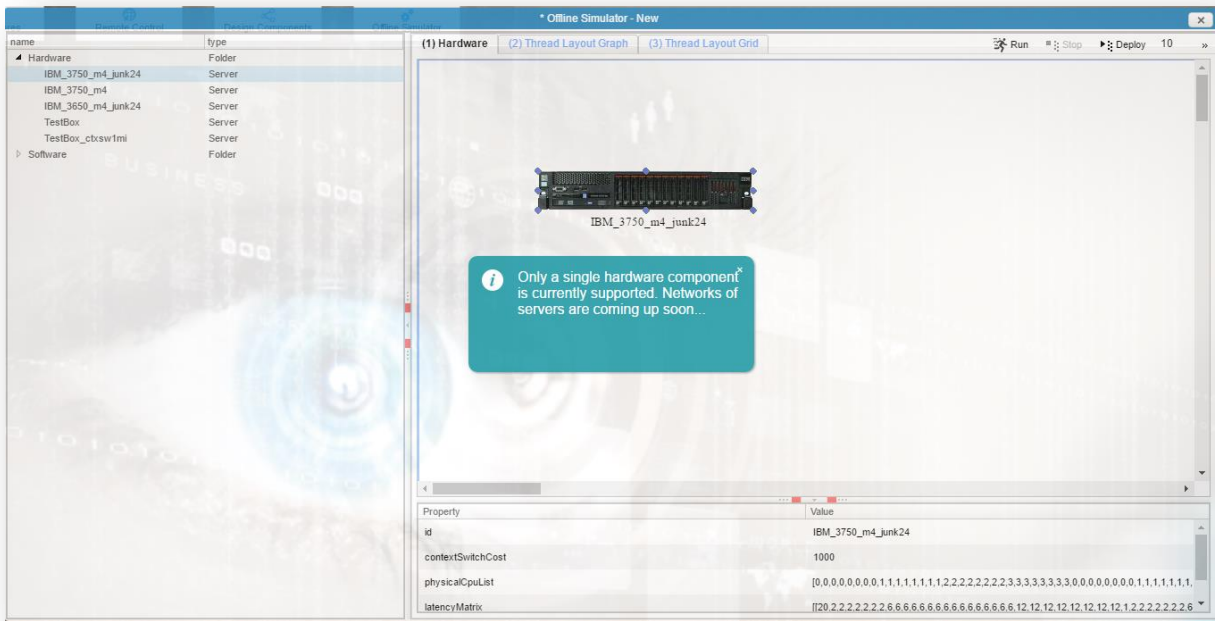
The Hardware Tab allows users to view the current hardware profile picked up by the relevant PVTM Agent, including the latest values for the following options that control simulation behaviour:

- pvtm_agent_link_proto_weights_un_i4_i6_ld_pf_lk_sm_mins_json_array
- pvtm_agent_link_proto_weights_un_i4_i6_ld_pf_lk_sm_maxs_json_array
- pvtm_agent_context_switch_cost

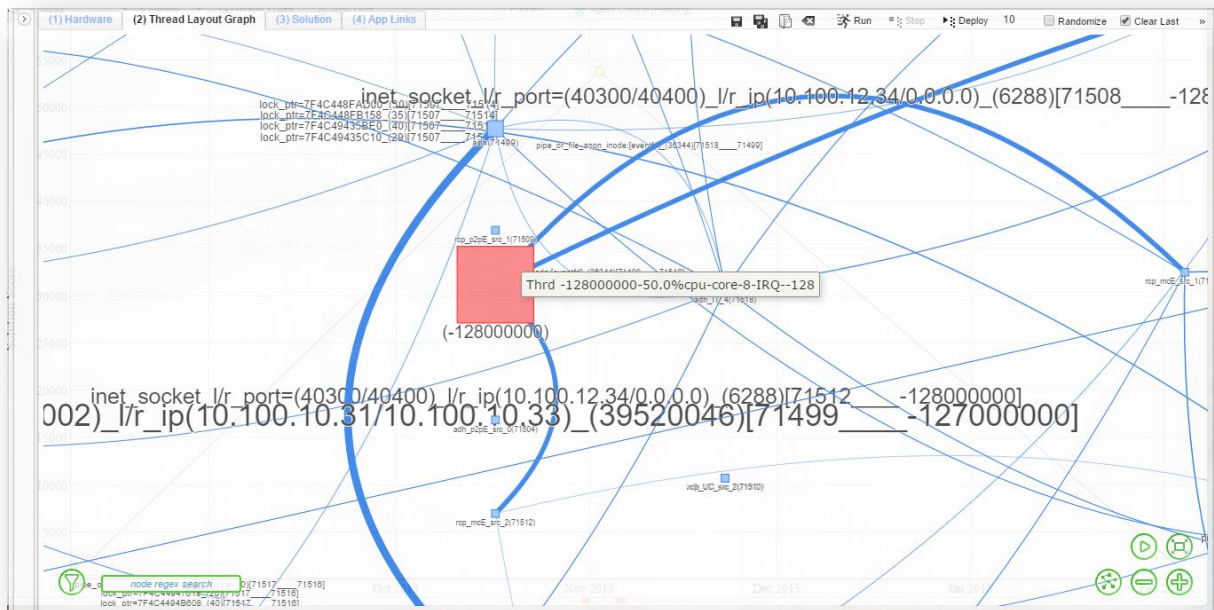
- pvtm_agent_junk_cores / pvtm_agent_junk_cores_json_array

Users can modify the values of some of these options and re-run ad-hoc simulations (by clicking on the Run button in the Thread Pinning Layout Toolbar) to experiment how these values impact the simulation results.

This view can also be used to simulate running the current software layout on different types of servers. This is extremely useful for capacity planning, as users can test the suitability of a new hardware platform, (including servers which are not yet available for sale), for applications before issuing a purchase order. To achieve this, users should expand the Hardware and Software Components Tree (if not already expanded), open the Hardware branch of the tree, and drag and drop one of the hardware entries into the canvas, and click on the Run button to re-run the simulation:



Thread Layout Tab



The Thread Layout tab allows users to introspect inter-processor communications links between software threads in an easy-to-understand graphical format. Squares represent application threads, and lines different types of inter-thread communication mechanisms, or App Links. The larger the size of the square, the higher is the CPU utilisation for that thread. The squares are colour-coded to match the application or process they belong to. **I**nterrupt **R**e**Q**uests (IRQs) are denominated as squares that have a negative Process ID. The thickness of the lines denotes the amount of data flowing between two threads through a particular App Link.

Views of very complex applications can, at first glance, look confusing, due to the sheer amount of lines and threads shown in the graph, but zooming in provides detailed information about the specific threads and links, including the following details:

Threads / IRQs






- CPU utilization
- thread id or interrupt number if a negative number
- thread name

App Links

- Type of IPC link, (socket, shared memory, named pipe, mutex, futex lock)
- Source and destination ports
- Data transfer in bytes

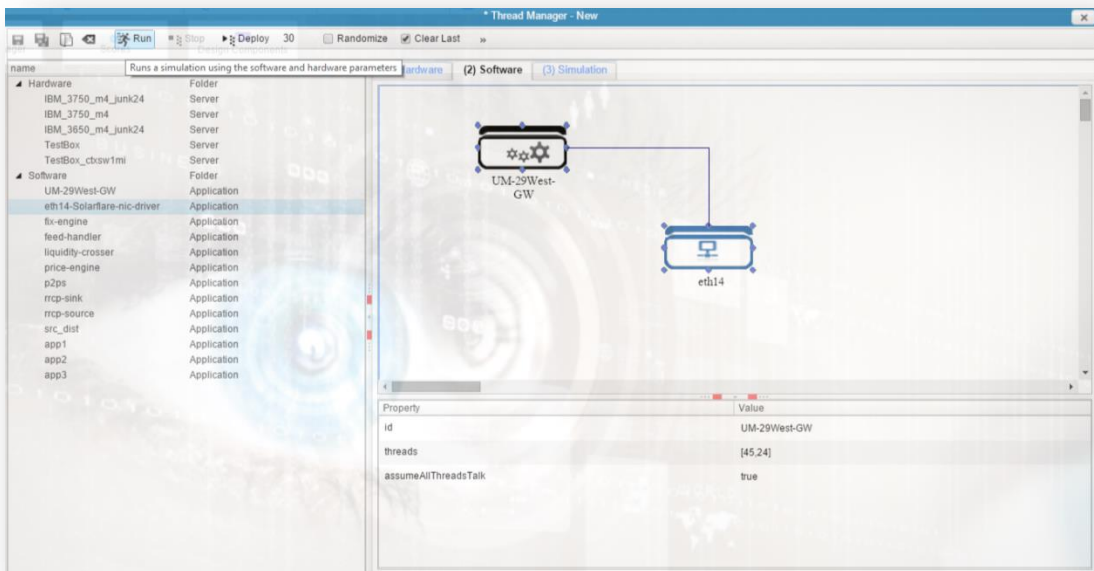
This tab can sometimes take a few seconds to render, especially if the system has a large number of threads and links. Clicking on the Pause button on the bottom right will pause the rendering engine and may speed up the display. Here is a quick description of how to navigate on this tab:

- Zoom in/out by scrolling or clicking on the  /  buttons

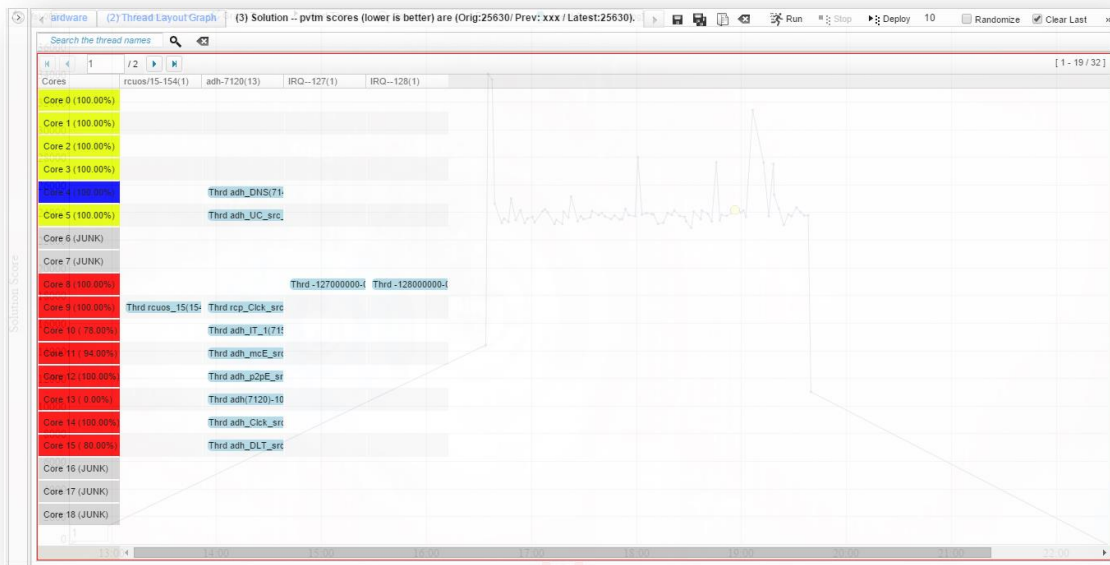
- Pause/play button stops  or starts  the physics-based animation
- Fit to size button  makes all threads appear on the screen
- Search textbox a regex that finds a thread by name or id, and zooms into its location
- Cosmos  or Hierarchical  mode buttons arrange the threads in a free-form cluster, or separate threads by the cores where they are running, aligning all threads running on the same core in rows or columns.

For more details, please see section titled "The Preview Area".

When starting the Offline Simulator Window in simulation mode (by clicking on the Thread Manager button in the Thread Manager tab in the Main Canvas's Navigation), the Software Tab appears differently, enabling users to drag and drop custom-built widgets (see The Design Components Window):





The Solution Tab



The Solution tab shows users a grid with the thread layout across the various cores in the system. The grid columns show each software process monitored by PVTM Agent. The grid Rows show the CPU cores of the hardware platform. The Threads are the light blue widgets inside the grid. By double-clicking on a core name, (highlighted in **BLUE**), users see which cores are closest to it, (highlighted in **GREEN**) and which are in remote NUMA zones, (highlighted in **AMBER** or **RED**). Double-clicking any core name again clears the colour coding.

Users are also able to drag and drop threads from one core to another. This results in an immediate change to the Score, and is valuable to developers as a sanity check of the thread layout.

Double-clicking on the application column headers forces the grid to only show cores where the application threads are currently allocated. Double clicking the heading again will show the whole grid again.

Users can also search for any string within the thread description by typing it in the textbox labelled 'Search the thread names', and clicking on the search button . The regex will try to match the text from any of the columns displayed. Clicking on the clear button  clears the filter, and displays the whole table again.

If PVTM Agent is running with root privileges, then a number of cores can be set aside as, "junk cores" (see `pvtm_agent_junk_cores`). These appear in grey with the word (JUNK) next to the core number. Junk cores are used to run applications that do not match the `pvtm_agent_regex_pattern`. These typically include PVTM Agent itself, monitoring software, and any other processes, that are not performance critical. This reduces interference with the key application/s being optimized.

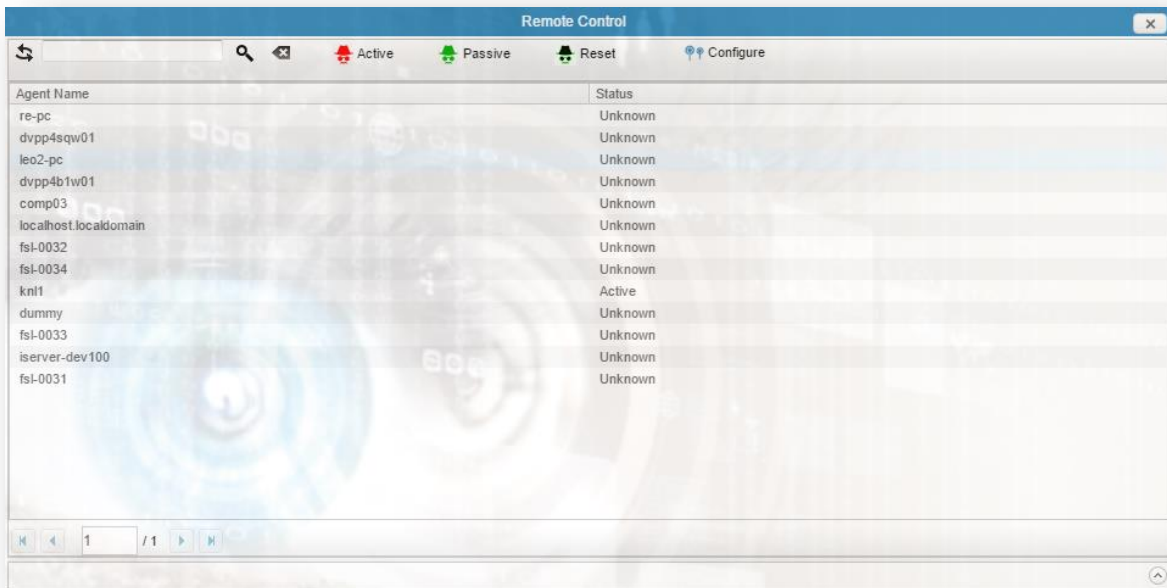
The App Links Tab

Source Thread	Target Thread	Weight	Full Text/Full Text
Thrd adh(71499)-6.0%cpu-core-8-adh-71499	Thrd -127000000-50.0%cpu-core-8-IRQ--127	80	inet_socket_wr_port=(60547/15002)_wr_ip(10.100.10.31/10.100.10.33)_(39520046/71499)____-127000000
Thrd adh(71499)-6.0%cpu-core-8-adh-71499	Thrd adh_DLT_src_1(71514)-8.0%cpu-core-1-adh-71499	5	pipe_or_file_anon_inode[eventfd](36872/71499)____71514
Thrd adh(71499)-6.0%cpu-core-8-adh-71499	Thrd adh_IT_1(71515)-2.0%cpu-core-13-adh-71499	4	pipe_or_file_anon_inode[eventfd](36848/71499)____71515
Thrd adh(71499)-6.0%cpu-core-8-adh-71499	Thrd adh_IT_2(71516)-2.0%cpu-core-15-adh-71499	4	pipe_or_file_anon_inode[eventfd](36208/71499)____71516
Thrd adh(71499)-6.0%cpu-core-8-adh-71499	Thrd adh_IT_3(71517)-4.0%cpu-core-0-adh-71499	1	lock_ptr=7F4C38FEDBD0_46/71499)____71516
Thrd adh(71499)-6.0%cpu-core-8-adh-71499	Thrd adh_IT_3(71517)-4.0%cpu-core-0-adh-71499	5	lock_ptr=7F4C36AFBAD0_47/71499)____71517
Thrd adh(71499)-6.0%cpu-core-8-adh-71499	Thrd adh_IT_4(71518)-2.0%cpu-core-9-adh-71499	4	pipe_or_file_anon_inode[eventfd](36344/71499)____71518
Thrd adh_Click_src_0(71502)-0.0%cpu-core-15-adh-71499	Thrd adh_mce_src_0(71503)-0.0%cpu-core-26-adh-71499	1	lock_ptr=7F4C49435670_8/71502)____71503
Thrd adh_Click_src_0(71502)-0.0%cpu-core-15-adh-71499	Thrd adh_mce_src_0(71503)-0.0%cpu-core-26-adh-71499	1	lock_ptr=7F4C49435670_8/71502)____71503
Thrd adh_Click_src_0(71502)-0.0%cpu-core-15-adh-71499	Thrd adh_mce_src_0(71503)-0.0%cpu-core-26-adh-71499	1	lock_ptr=7F4C494357C0_13/71502)____71503
Thrd adh_Click_src_0(71502)-0.0%cpu-core-15-adh-71499	Thrd adh_mce_src_0(71503)-0.0%cpu-core-26-adh-71499	1	lock_ptr=7F4C494358A58_7/71502)____71503
Thrd adh_Click_src_0(71502)-0.0%cpu-core-15-adh-71499	Thrd adh_mce_src_0(71503)-0.0%cpu-core-26-adh-71499	1	lock_ptr=7F4C4946D440_4/71502)____71503
Thrd adh_mce_src_0(71503)-0.0%cpu-core-26-adh-71499	Thrd -128000000-50.0%cpu-core-8-IRQ--128	50	inet_socket_wr_port=(40300/40400)_wr_ip(10.100.12.34/0.0.0.0)_(6288/71503)____-128000000
Thrd adh_Click_src_0(71502)-0.0%cpu-core-15-adh-71499	Thrd adh_Click_src_0(71502)-0.0%cpu-core-15-adh-71499	1	lock_ptr=7F4C49435640_7/71502)____71502
Thrd adh_mce_src_0(71503)-0.0%cpu-core-26-adh-71499	Thrd adh_Click_src_0(71502)-0.0%cpu-core-15-adh-71499	1	lock_ptr=7F4C49435670_8/71502)____71502
Thrd adh_mce_src_0(71503)-0.0%cpu-core-26-adh-71499	Thrd adh_Click_src_0(71502)-0.0%cpu-core-15-adh-71499	1	lock_ptr=7F4C49435730_13/71502)____71502
Thrd adh_mce_src_0(71503)-0.0%cpu-core-26-adh-71499	Thrd adh_Click_src_0(71502)-0.0%cpu-core-15-adh-71499	1	lock_ptr=7F4C494357C0_13/71502)____71502
Thrd adh_mce_src_0(71503)-0.0%cpu-core-26-adh-71499	Thrd adh_Click_src_0(71502)-0.0%cpu-core-15-adh-71499	1	lock_ptr=7F4C494358A58_7/71502)____71502
Thrd adh_mce_src_0(71503)-0.0%cpu-core-26-adh-71499	Thrd adh_Click_src_0(71502)-0.0%cpu-core-15-adh-71499	1	lock_ptr=7F4C4946D440_4/71502)____71502
Thrd rcp_Click_src_1(71507)-0.0%cpu-core-14-adh-71499	Thrd rcp_mce_src_1(71508)-0.0%cpu-core-11-adh-71499	3	lock_ptr=7F4C448FAD00_27/71507)____71508
Thrd rcp_Click_src_1(71507)-0.0%cpu-core-14-adh-71499	Thrd rcp_mce_src_1(71508)-0.0%cpu-core-11-adh-71499	3	lock_ptr=7F4C448FB158_26/71507)____71508
Thrd rcp_Click_src_1(71507)-0.0%cpu-core-14-adh-71499	Thrd rcp_mce_src_1(71508)-0.0%cpu-core-11-adh-71499	3	lock_ptr=7F4C49435BE0_26/71507)____71508
Thrd rcp_Click_src_1(71507)-0.0%cpu-core-14-adh-71499	Thrd rcp_mce_src_1(71508)-0.0%cpu-core-11-adh-71499	3	lock_ptr=7F4C49435C10_26/71507)____71508
Thrd rcp_Click_src_1(71507)-0.0%cpu-core-14-adh-71499	Thrd adh_DLT_src_1(71514)-8.0%cpu-core-1-adh-71499	3	lock_ptr=7F4C448FAD00_30/71507)____71514
Thrd rcp_Click_src_1(71507)-0.0%cpu-core-14-adh-71499	Thrd adh_DLT_src_1(71514)-8.0%cpu-core-1-adh-71499	3	lock_ptr=7F4C448FB158_35/71507)____71514
Thrd rcp_Click_src_1(71507)-0.0%cpu-core-14-adh-71499	Thrd adh_DLT_src_1(71514)-8.0%cpu-core-1-adh-71499	4	lock_ptr=7F4C49435BE0_40/71507)____71514
Thrd rcp_Click_src_1(71507)-0.0%cpu-core-14-adh-71499	Thrd adh_DLT_src_1(71514)-8.0%cpu-core-1-adh-71499	3	lock_ptr=7F4C49435C10_29/71507)____71514
Thrd rcp_mce_src_1(71508)-0.0%cpu-core-11-adh-71499	Thrd -128000000-50.0%cpu-core-8-IRQ--128	50	inet_socket_wr_port=(40300/40400)_wr_ip(10.100.12.34/0.0.0.0)_(6288/71508)____-128000000

The App Links tab, provides the same information as the Thread Layout Tab, but in an easy-to-read tabular format. It allows users to filter out unnecessary links or software threads by typing regex strings in the top left text box labelled 'Search the Links', and clicking on the search button . The regex will try to match the text from any of the columns displayed. Clicking on the clear button clears the filter, and displays the whole table again. Users can also click on the column headings to sort the table.

When starting the Offline Simulator Window in simulation mode (by clicking on the Thread Manager button in the Thread Manager tab in the Main Canvas's Navigation), the App Links tab currently does not appear.

The Remote Control Window

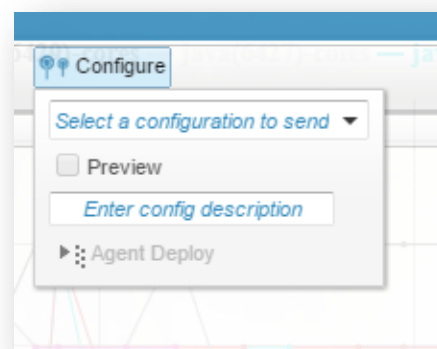


This window enables users to select a set of agents, query their status (using the refresh button ↻), or activate 🛑 / deactivate 🟢 / reset 🟢 PVTM Agents dynamically. In addition, the Configure 🛠️ Button enables configurations to be pushed to multiple PVTM Agents. The search button 🔍 uses the regex filter in the textbox to its left, whereas the clear button 🗑️ clears out the filter. Only the current list of agents (filtered or otherwise) will be affected by the activate/deactivate/reset/configure commands. Inactive agents never automatically apply thread pinning strategies.

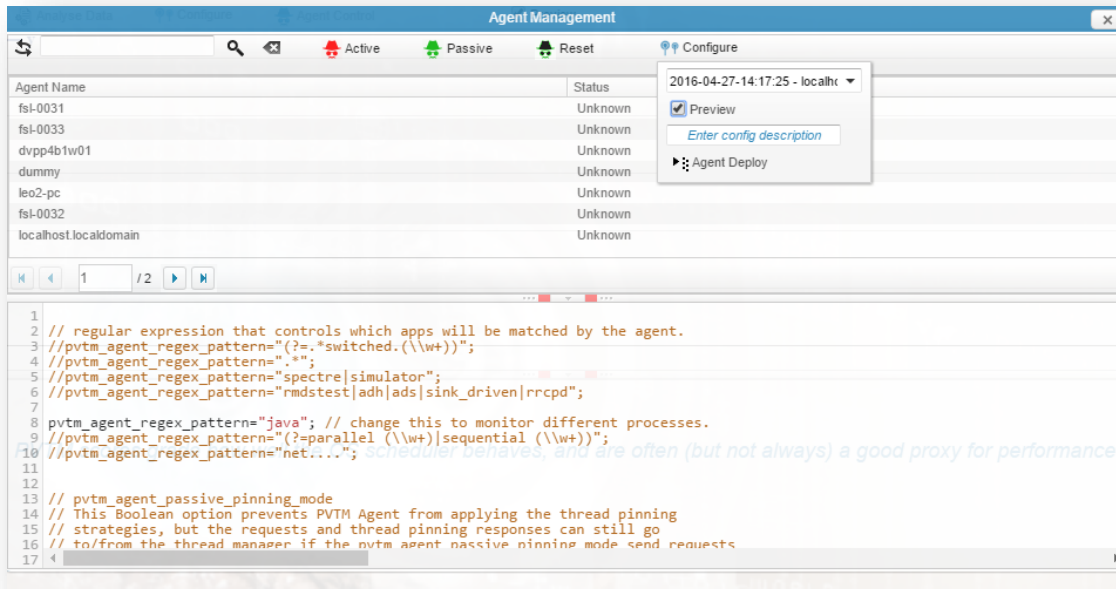
The Configure Menu

Clicking on the Configure 🛠️ Button brings up a pop-up menu similar to the one on the right with the following items:

- Select a configuration to send - This combo box shows the last 1024 configurations that were sent to PVTM Agents.
- The Preview checkbox shows a preview of the configuration in the bottom half of the PVTM Agent Control window.
- The Enter config description textbox allows users to enter a description for the changes being made.
- The Agent Deploy button pushes the configuration to the current list of agents (filtered or otherwise).

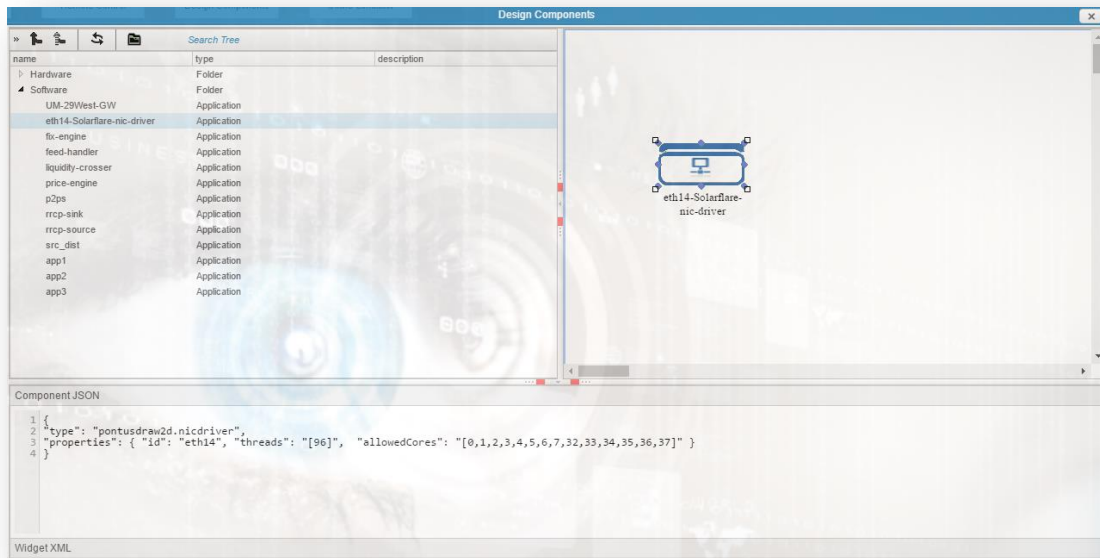


The Configuration Preview



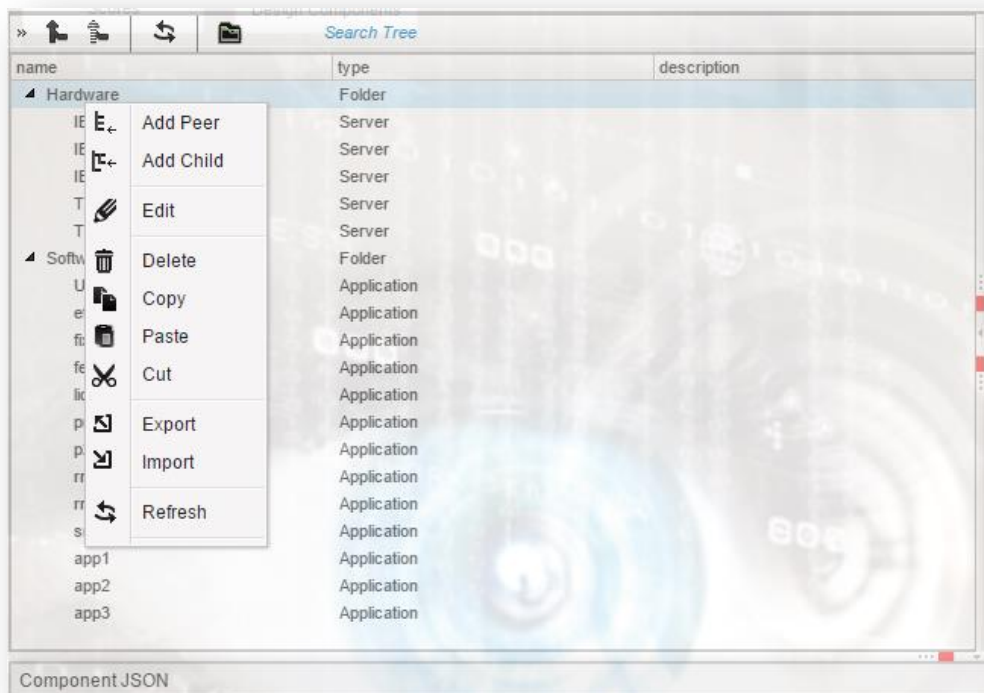
The configuration preview panel appears when the Configure Preview checkbox is checked. This enables users to edit the configuration before sending it out to the current list of agents in the PVTM Agent grid above (filtered or otherwise). Once done editing, users can click on the Agent Deploy button to deploy the existing configuration. Note that this will cause PVTM Agents to disconnect and connect again.

The Design Components Window



The Widgets Window is composed of 3 elements: The Hardware and Software Components Tree, the Component JSON Editor, and the Preview Canvas.

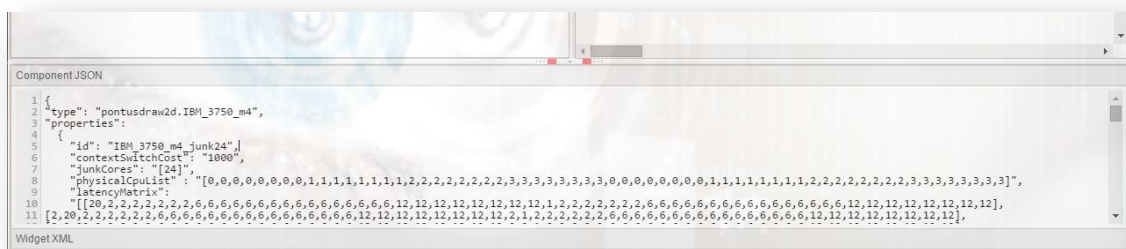
The Hardware and Software Components Tree



The Hardware and Software Components tree allows users to create new hardware or software widgets that are available for users to use in the Hardware and Software Components Tree in The Offline Simulator Window. This element has the following parts:

- Hardware Components Tree – allows users to create new hardware components by right clicking and selecting 'add child' or 'add peer'.
- Software Components Tree – allows users to create new software components by right clicking and selecting 'add child' or 'add peer'.
- Menu/Toolbar allows users to manipulate the Component Trees:
 - Edit Menu – allows users to change the component name
 - Delete deletes a component
 - Copy/Paste do as they say
 - Export - allows users to export the components to an external file
 - Import - allows components to be imported back in.....

Component JSON Editor

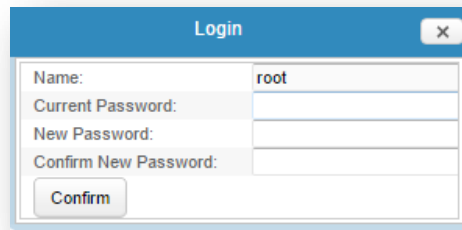


The Component JSON Editor allows users to enter the JSON definition of either software or hardware components (see SET_COMPUTER and SET_APPS for more details). To use this, users must click on an existing software or hardware component from The Hardware and Software Components Tree.


Preview Canvas

The Preview Canvas is located to the right of The Hardware and Software Components Tree allows users to drag and drop components from the Hardware and Software Components Tree to preview the icons. Note that currently, only built-in icons are supported; the icons are derived from the "Type" field in the Component JSON definition.

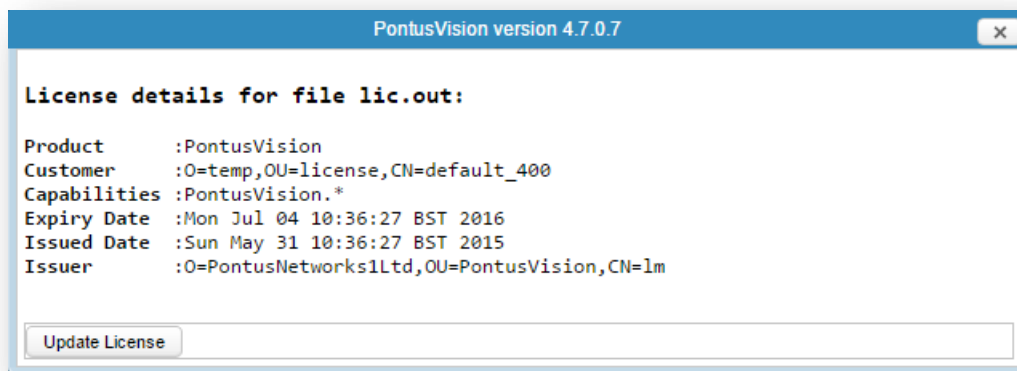
Changing the Password



A dialog box titled "Login" with a close button (X) in the top right corner. It contains four input fields: "Name:" with the value "root", "Current Password:", "New Password:", and "Confirm New Password:". A "Confirm" button is located at the bottom left of the dialog.

To change the current user's password, click on the Top Toolbar's Users Icon . That will open the window above prompting for the current password, as well as the new one. To cancel this, click on the X on the top right corner.


Updating the License



A dialog box titled "PontusVision version 4.7.0.7" with a close button (X) in the top right corner. The main content area displays the following text:

```
License details for file lic.out:  
  
Product      :PontusVision  
Customer     :O=temp,OU=license,CN=default_400  
Capabilities :PontusVision.*  
Expiry Date  :Mon Jul 04 10:36:27 BST 2016  
Issued Date  :Sun May 31 10:36:27 BST 2015  
Issuer       :O=PontusNetworks1Ltd,OU=PontusVision,CN=lm
```

An "Update License" button is located at the bottom left of the dialog.

To check the current license's details, and to update the license key, users can click on the the Top Toolbar's Information Icon . That will open the window above showing the current license information. Clicking on the Update License button brings up a file manager-style dialogue box prompting the user to select a new license file.

Command Line

PVTM GUI uses the same jar file as the PVTM Simulator. The command line is under the <installation directory>/4.7.0/ folder. The windows version is in the run-gui.bat file, whereas the Linux version is in the run-gui.sh file. Normally, the installer fills out the contents of these files, but in certain circumstances, they need to be manually edited. Here is a list of the main command line arguments:

- `-Xmx2048m`
This controls the maximum memory size of the Java Virtual Machine; this value should be approximately 500MB-1GB per agent being monitored.
- `-javaagent:/home/leo/pontus-vision/4.7.0/linux/pvtm-agent.jar`
This optional command line argument exposes java thread names to the OS (on Linux only)
- `-Dpv.threadmgr.configFile=""`
This option allows an optional configuration file to be passed to PVTM Simulator to control the simulator's behaviour; this option should only be used after engaging with Pontus's professional services team.
- `-Dpontusvision.threads.in.levels=true`
This option determines whether or not to display the threads that belong to the same application in levels (see Thread Layout Tab for more details)
- `-Dpontusvision.clean_background.enabled=false`
This option enables or disables the background picture with the big 'PontusVision Eye'. Setting this to true uses a clear white background. This can be useful if running the browser pointing to the PVTM GUI in a remote desktop with low bandwidth.
- `-Dpontusvision.static_web_files_location="C:\Users\L\pontus-vision\4.7.0\web"`
This option determines the location of web files serviced by the clear-text http_port
- `-Dpontusvision.keystore="C:\Users\L\.keystore_pontusvision"`
This option determines the location of the key store used by the PVTM GUI to encrypt https connections
- `-Dpontusvision.http_port=8080`
This option determines the clear port used by the PVTM GUI to service static files
- `-Dpontusvision.https_port=8443`
This option determines the encrypted port used when accessing the PVTM GUI from a browser (e.g. `https://<server>:<pontusvision.https_port>`)
- `-Dpontusvision.data.jdbc.url="jdbc:h2:C:\Users\L\pontusvision"`
This option determines the URL for the static Database files used by the PontusVision framework to store templates and saved thread pinning strategies.
- `-Dpontusvision.audit.jdbc.url="jdbc:h2:C:\Users\L\auditDb"`
This option determines the URL for the static Database files used to maintain an audit trail of actions in the PVTM GUI.
- `-Dpontusvision.reindex_text_search=true`
This option determines whether or not the PVTM GUI should re-index all the stored template and attributes text so the search functionality works.
- `-Dpontusvision.license.info="PontusVision.*"`
This option selects the license information for PVTM Simulator. This value is customer-specific, and is typically entered as part of the installation process.
- `-Dpontusvision.license.holder="o=temp,ou=license,cn=test123"`
This option selects the license key information for PVTM Simulator. This value is customer-specific, and is typically entered as part of the installation process.

- `-Dpontusvision.license.file=~/.pontus-vision/4.7.0/license.lic`
This option selects the license file information for PVTM Simulator. This value is customer-specific, and is typically entered as part of the installation process.
- `-Dpontusvision.start_browser=true`
This option determines whether or not PVTM GUI starts a new instance of the default browser on Windows
- `-Dpontusvision.data.replication_mode=""`
This option determines the replication mode for the static Database used by PVTM GUI. An empty string means standalone, whereas "primary" indicates a master instance running in primary mode, and "backup" indicates a slave instance running in backup mode.
- `-Dpontusvision.data.replication_url="tcp://localhost:61616"`
This is the JMS URL used for the data replication between primary and backup instances.
- `-Dpontus.agent.broker.name="tcp://localhost:61616"`
This option controls the JMS Broker name used by PVTM Simulator to communicate with PVTM GUI and send/receive dynamic commands to PVTM Agent. NOTE: This option is also used as the JMS URL for the PontusVision Agent (NOT to be confused with the PVTM Agent) to communicate with the GUI. The PontusVision Agent is only used to deploy configuration files and start/stop processes remotely. It is a part of the PontusVision framework, and is NOT REQUIRED for PVTM.
- `-Dpv.threadmgr.to.gui.topic="pvtm.threadmgr.to.gui.topic"`
This option controls the JMS Topic used to send data from PVTM Simulator to PVTM GUI.
- `-Dpv.threadmgr.to.gui.topic="pvtm.threadmgr.from.gui.topic"`
This option controls the JMS Topic used to send data from PVTM GUI to PVTM Simulator.
- `-Dpontusvision.bpmn.enabled=true`
This option turns on certain aspects of the Thread Management GUI. Do not touch this option.
- `-Dpvtm_history_prefix="/home/leo/pontus-vision/4.5.0/pvtm-storage/"`
This option selects the folder where the embedded Elastic Search engine will store its data. This should be a local disk with as much storage as possible.
- `-jar "C:\Users\L\pontus-vision\4.7.0\vision-server.jar"`
This option controls the name of the jar file used to start the PVTM GUI. Note that this is the same jar file as the PVTM Simulator.