The HOPSA Workflow and Tools

Bernd Mohr, Vladimir Voevodin, Judit Giménez, Erik Hagersten,
Andreas Knüpfer, Dmitry A. Nikitenko, Mats Nilsson, Harald Servat, Aamer Shah,
Frank Winkler, Felix Wolf, and Ilya Zhujov

Abstract To maximise the scientific output of a high-performance computing system, different stakeholders pursue different strategies. While individual application developers are trying to shorten the time to solution by optimising their codes, system administrators are tuning the configuration of the overall system to increase its throughput. Yet, the complexity of today's machines with their strong interrelationship between application and system performance presents serious challenges to achieving these goals. The HOPSA project (HOlistic Performance System Analysis) therefore sets out to create an integrated diagnostic infrastructure for combined application and system-level tuning - with the former provided by the EU and the latter by the Russian project partners. Starting from system-wide basic performance screening of individual jobs, an automated workflow routes findings on potential bottlenecks either to application developers or system administrators with recommendations on how to identify their root cause using more powerful diagnostic tools. Developers can choose from a variety of mature performance-analysis

Bernd Mohr, Ilya Zhukov
Forschungszentrum Jülich GmbH, Jülich Supercomputing Centre, Germany,
e-mail: `{b.mohr,i.zhukov}@fz-juelich.de`

Vladimir Voevodin, Dmitry A. Nikitenko
Moscow State University, RCC, Russia, e-mail: `{voevodin,dan}@parallel.ru`

Judit Giménez, Harald Servat
Barcelona Supercomputing Centre, Spain, e-mail: `{judit,harald.servat}@bsc.es`

Felix Wolf, Aamer Shah
German Research School for Simulation Sciences GmbH / RWTH Aachen University, Germany,
e-mail: `{f.wolf,a.shah}@grs-sim.de`

Erik Hagersten, Mats Nilsson
Rogue Wave Software AB, Sweden,
e-mail: `{Erik.Hagersten,Mats.Nilsson}@roguewave.com`

Andreas Knüpfer, Frank Winkler
Technical University Dresden, Germany,
e-mail: `{andreas.knuepfer,frank.winkler}@tu-dresden.de`

tools developed by our consortium. Within this project, the tools will be further integrated and enhanced with respect to scalability, depth of analysis, and support for asynchronous tasking, a node-level paradigm playing an increasingly important role in hybrid programs on emerging hierarchical and heterogeneous systems.

1 Introduction

To maximise the scientific and commercial output of a high-performance computing system, different stakeholders pursue different strategies. While individual application developers are trying to shorten the time to solution by optimising their codes, system administrators are tuning the configuration of the overall system to increase its throughput. Yet, the complexity of today's machines with their strong interrelationship between application and system-level performance demands an integration of application and system programming.

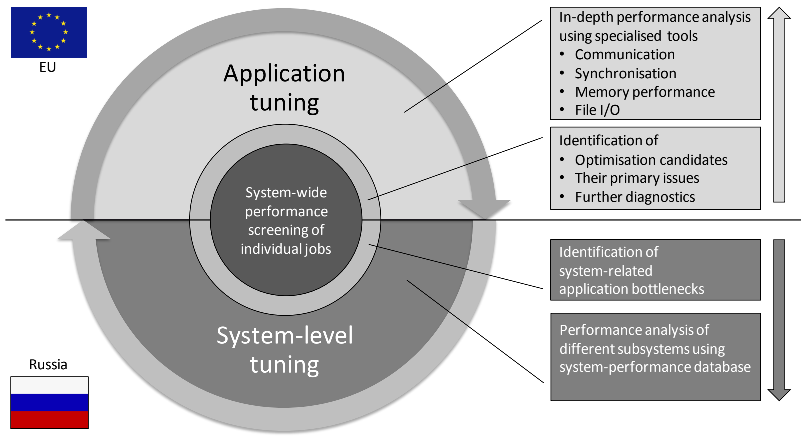

EU
Application tuning
System-wide performance screening of individual jobs
System-level tuning
Russia
In-depth performance analysis using specialised tools
• Communication
• Synchronisation
• Memory performance
• File I/O
Identification of
• Optimisation candidates
• Their primary issues
• Further diagnostics
Identification of system-related application bottlenecks
Performance analysis of different subsystems using system-performance database

Fig. 1 System-level tuning (bottom), application-level tuning (top), and system-wide performance screening (centre) use common interfaces for exchanging performance properties.

The HOPSA project (HOlistic Performance System Analysis) therefore sets out for the first time to develop an integrated diagnostic infrastructure for combined application and system-level tuning. Using more powerful diagnostic tools application developers and system administrators will easier identify the root causes of their respective bottlenecks. With the HOPSA infrastructure, it is more effective to optimise codes running on HPC systems. More efficient codes mean either getting results faster or being able to get higher quality or more results in the same time.

The work in HOPSA is carried out by two coordinated projects funded by the EU under call FP7-ICT- 2011-EU-Russia and the Russian Ministry of Education and Science, respectively. Its objective is the new innovative integration of application tuning with overall system diagnosis and tuning to maximise the scientific output of our HPC infrastructures. While the Russian consortium will focus on the system aspect, the EU consortium will focus on the application aspect. At the interface between these two facets of our holistic approach, which is illustrated in Figure 1, is the system-wide performance screening of individual jobs, pointing at both inefficiencies of individual applications and system-related performance issues.

This article describes the overall workflow of performance analysis of parallel programs using the HOPSA infrastructure, introduces the individual tools developed inside the project consortium, and shows how to use the tools in a complementary way. For detailed information, please see the user guides of the individual performance-analysis tools.

At the centre of this workflow is the so-called lightweight measurement module (LWM^2). It is responsible for the first step in the workflow, the system-wide mandatory collection of basic performance data. For each execution on the cluster, LWM^2 produces a so-called job digest. The metrics listed in this compact report indicate whether an application suffers from an inherent performance problem or whether application interference may have been at the root of dissatisfactory behaviour. They also provide a first assessment regarding the nature of a potential performance problem and help to decide on further diagnostic steps using any of the more powerful performance-analysis tools. For each of those tools, a short summary is given with information on the most important questions it can help to answer. Moreover, the document covers Score-P [5], a common measurement infrastructure shared by some of the tools. The performance data types supported by Score-P form a natural refinement hierarchy that can be followed to track down and represent even complex bottleneck situations at increasing levels of granularity. Finally, a brief excursion on system-level tuning explains how system providers can leverage the data collected by LWM^2 to identify a suboptimal system configuration or faulty components.

2 The HOPSA Workflow

2.1 Overview

The performance-analysis workflow (Figure 2) consists of two basic steps. During the first step, we identify all those applications running on the system that may suffer from inefficiencies. This is done via system-wide job screening supported by a lightweight measurement module (LWM^2) dynamically linked to every executable. The screening output identifies potential problem areas such as communication, memory, or file I/O, and issues recommendations on which diagnostic tools can be used to explore the issue further. Available application performance analysis tools include Paraver/Dimemas [1, 12], Scalasca [2], ThreadSpotter [3], and Vampir [4]. The data collected by LWM^2 is also fed into the Clustrx.Watch hierarchical cluster monitoring system [13] which combines it with system and hardware data and forwards it to the LAPTA cluster monitoring and analysis system [14] for further analysis by system administrators.

In general, the workflow successively narrows the analysis focus and increases the level of detail at which performance data are collected. At the same time, the measurement configuration is optimised to keep intrusion low and limit the amount of data that needs to be stored. To distinguish between system and application-related performance problems, some of the tools allow also system-level data to be retrieved and displayed. The system administrator, in contrast, has access to global performance data. He can use this data to identify potential system performance bottlenecks and to optimise the system configuration based on current workload needs. In addition, the administrator can identify applications that continuously underperform and proactively offer performance-consulting services. In this way, it becomes possible to reduce the unnecessary waste of expensive system resources.

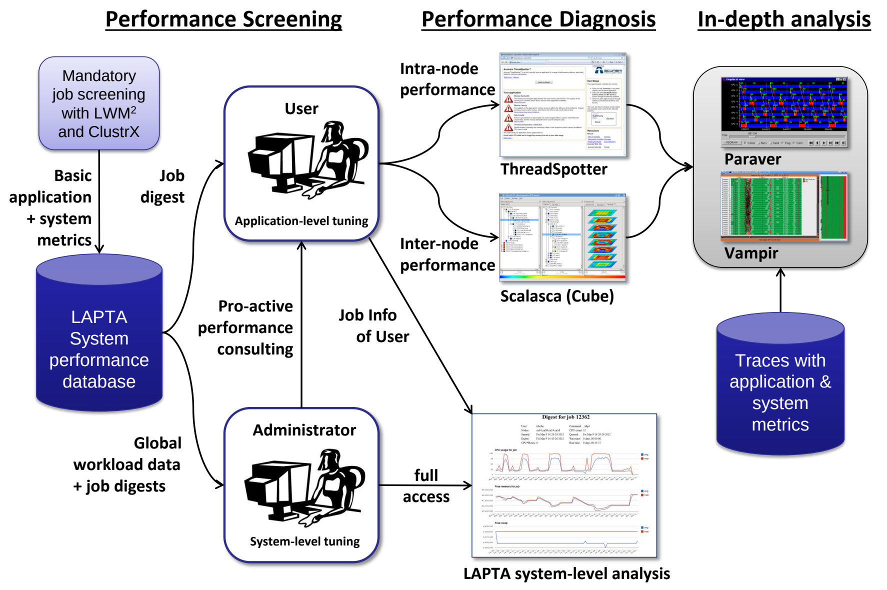

Performance Screening
Performance Diagnosis
In-depth analysis
Mandatory job screening with LWM² and ClustrX
Basic application + system metrics
Job digest
User
Application-level tuning
Intra-node performance
ThreadSpotter
Inter-node performance
Scalasca (Cube)
Paraver
Vampir
LAPTA System performance database
Pro-active performance consulting
Job Info of User
Traces with application & system metrics
Global workload data + job digests
Administrator
System-level tuning
full access
LAPTA system-level analysis

Fig. 2 Overview of the performance analysis workflow.

2.2 *Performance Screening*

This step decides whether an application behaves inefficiently. On the side of the user, nothing has to be done except running the application as usual. Upon application start, LWM^2 is automatically and transparently linked to the executable through library pre-loading. At runtime, the module collects basic performance data with very low overhead. The performance data characterise various aspects such as sequential performance, parallel performance, and file I/O. At the end of execution, the user receives a job digest that contains the most important performance metrics. The digest also recommends further diagnostics in the case certain key metrics show unexpected values, which may often be indicative of a performance problem. If needed, the user can disable LWM^2, for example, to avoid interference with the analysis tools used in subsequent stages of the tuning process.

2.2.1 The Lightweight Measurement Module LWM^2

The lightweight measurement module LWM^2 collects basic performance data for every process of a parallel application. It supports applications based on MPI and multithreaded applications based on POSIX Threads or any higher-level model implemented on top of it, which usually includes OpenMP. Multithreaded MPI applications and applications that additionally use CUDA are supported as well. To keep the overhead at a minimum, the module applies a combination of sampling and careful direct instrumentation via interposition wrappers. Direct instrumentation is needed to track the state of a thread (e.g., whether it executes inside or outside an MPI function) and to access relevant communication or I/O parameters such as the number of bytes sent or written to disk. Based on the state tracking performed by the instrumentation, sampling partitions the execution time into different components such as computation, communication, or I/O. LWM^2 refrains from direct time measurements as far as possible. Hardware counters deliver basic information on single-node performance. To save storage space, the performance data of individual threads are folded into per-process metrics such as the average number of threads.

In addition to collecting performance data separately for each process, LWM^2 divides the time axis into disjoint slices, recording selected metrics related to the use of shared resources at this finer granularity. The slices have a length of 10s and are synchronized across the entire machine. Together with the location of each process on the cluster, which LWM^2 records along with the performance data, LWM^2 provides performance data for each active cell of a cluster-wide time-space grid. The discretised time axis constitutes the first dimension, the nodes of the system the second one.

The purpose of organising the performance data in this way is threefold: First, by comparing the data of different jobs that were active during the same time slice, it becomes possible to see signs of interference between applications. Examples include reduced communication performance due to overall network saturation or low I/O bandwidth due to concurrent I/O requests from other jobs. Second, by looking at the performance data of the same node across a larger number of jobs and comparing it to the performance of other nodes during the same period, anomalies can be detected that would otherwise be hidden when analysing performance data only on a per-job basis. Third, collecting synchronised performance data from all the jobs running on a given system will open the way for new directions in the development of job scheduling algorithms that take the performance characteristics of individual jobs into account. For example, to avoid file-server contention and waiting time that may occur in its wake, it might be wiser not to co-schedule I/O-intensive applications. In this way, overall system utilisation may be further improved.

After the expiration of every time slice, LWM^2 passes the data of the current time slice on to Clustrx.Watch, a system-monitoring infrastructure running on each node. Clustrx augments these data with system data collected using various sensors and forwards them to the LAPTA system performance database.

2.2.2 LAPTA

LAPTA is a pAckage for Performance moniToring and Analysis. The software is aimed at providing flexible, scalable and extendable infrastructure for system-level performance analysis. It includes special tools and interfaces for data collection supporting various data collectors (Clustrx, Ganglia, LWM^2, etc.), data storage supporting wide range of databases (MongoDB, Cassandra, etc.) and both stored and streamed data access and analysis. LAPTA provides interfaces to access the collected system monitoring data for both query models: post mortem and on-the-fly. For example LAPTA serves as the basis for Job Digest generation based on system-level performance monitoring data. The screening of general job behavior through Job Digest is very useful for users and tuners to understand the possible bottlenecks that can be seen at a glance (like network overload, bad data locality, inefficient memory usage, too intensive I/O, etc.). Also, performance data of the same application collected over an extended period of time will document the tuning and scaling history of this application allowing to make even more detailed analysis of the dynamic application behavior further. Studying the performance behaviour of the entire job mix will allow to make conclusions on the optimal system configuration for the given workload. For example, system providers will learn whether requirements to amount of physical memory available, I/O or network bandwidth and other system hardware requirements were over- or underestimated.

2.3 Performance Diagnosis

This step decides why an application behaves inefficiently. It is only needed if the screening identifies a potential performance problem. Depending on the recommendation made by LWM^2 , the user chooses one or more of the performance-analysis tools offered by the HOPSA tool environment. The general strategy of the diagnosis is to start with an overview and then to go deeper as more information on the problem's root cause becomes available.

2.3.1 Overview of the Performance Analysis Tool Suite

An overview of the HOPSA performance analysis tool suite is presented in Table 1.

For the analysis of intra-node performance, ThreadSpotter is the primary tool, with the possibility of more detailed analyses using Paraver. For investigating inter-

Table 1 Classification of tools based on problem class and level of detail.

	Intra-node peformance	Inter-node peformance	I/O
Overview	ThreadSpotter	Score-P Profile + Cube	Scalasca(Cube)
In-depth analysis	ThreadSpotter, Paraver, Vampir	Scalasca Trace Analyzer + Cube, Paraver, Vampir	Paraver, Vampir

node performance, looking at a performance profile using Scalasca's Cube browser is a good starting point. For even more detailed analyses, the results of the Scalasca trace-analyser can be displayed in Cube, or the Vampir and Paraver/Dimemas tools can be used for a detailed visual exploration of the traces. For understanding I/O-related issues, profiles displayed in the Cube browser give a good overview, while Vampir can be used for more in- depth analysis.

2.3.2 The Score-P Instrumentation and Measurement System

The Score-P [5] measurement infrastructure is a highly scalable and easy-to-use tool suite for profiling, event tracing, and online analysis of HPC applications. It collects performance data that can be analysed using the HOPSA tools Scalasca and Vampir. In addition, it supports the performance tools Persicope [6] and TAU [7] developed outside the HOPSA project. Score-P has been created in the projects SILC and PRIMA funded by the German Ministry of Education and Research and the US Department of Energy, respectively. It will be maintained and further enhanced in a number of follow-up projects including HOPSA.

The main performance data formats produced by Score-P are CUBE-4 [8] for profiles and OTF2 [9] for event traces. Profiles provide a compact performance overview, while event traces allow the in-depth analysis of parallel performance phenomena. While classic profiles aggregate performance metrics across the entire execution, time-series profiles treat individual iterations of the application's main loop separately, which allows studying the temporal evolution of the performance behaviour. They provide less detail than event traces, but can cover longer executions. Together, the above-mentioned options form a hierarchy of performance data types with increasing level of detail. The main advantage of Score-P is that a user needs to become familiar with only one set of instrumentation commands to produce all theses data types, which can be analysed using the majority of the tools listed Table 1. Figure 3 provides and overview of the different performance data types supported by Score-P and the tools that can be used to analyse them. Below we cover the individual data types in more detail.

Profiles Profiles in the CUBE-4 format map a set of performance metrics such as the time spent on some activity or the number of messages sent or received onto pairs of call paths and processes (or threads in multithreaded applications).

Metrics with a specialization (i.e., subset) relationship can be arranged and displayed in a hierarchy. The call-path dimension forms the natural call-tree hierarchy. Processes and threads are also arranged in an inclusion hierarchy together with hardware components such as the nodes they reside on. In addition, it is possible to define Cartesian process topologies to represent network or virtual topologies. Profiles can be visually explored using the Cube browser. Compared to its predecessor CUBE-3, CUBE-4 files have been optimized for fast writing by storing the metric values in a binary file.

Time-Series Profiles Time-series profiles are like normal CUBE-4 profiles except that they maintain a separate sub-tree in the call tree for each iteration of the time-step loop. This allows the user to distinguish individual iterations and to observe the evolution of the performance behaviour along the time axis. Time-series profiles are created by annotating the body of the time-step loop with special instrumentation, which tells Score-P when an iteration ends and when a new one begins. They can be analysed using the normal Cube display. A future version of Cube (to be completed after this project ends) will provide special iteration diagrams that offer an easy way to judge how the performance changes over time. To avoid that profiling data exceeds the available buffer space, future versions of Score-P will support the dynamic compression of time-series profile data using an online clustering algorithm [15].

Event Traces Event traces include all events of an application run that are of interest for later examination, together with the time they occurred and a number of event-type-specific attributes. Typical events are entering and leaving of functions or sending and receiving of messages. Event traces produced by Score-P are stored in the Open Trace Format Version 2 (OTF-2), a new trace format whose design is based on the experiences with the two predecessor formats OTF [10] and EPILOG [11], the former native formats of Vampir and Scalasca, respectively. The main characteristics of OTF-2 are similar to other record-based parallel event trace formats. It contains events and definitions and distributes data storage over multiple files. In addition, it is more memory efficient, offering the possibility to achieve measurements with less perturbation due to memory buffer flushes. In contrast to OTF, the event traces are stored in a binary format, which reduces the size of the trace files without the need for a separate compression step. OFT-2 traces are the foundation for further analysis. Vampir can display OTF-2 traces visually using different kinds of displays, including a zoomable timeline. The Scalasca trace analyser identifies wait states and their root causes, producing a CUBE-4 file that provides a higher-level view of the application performance data. This is typically recommended to get an idea of key performance issues before visually exploring the traces directly using a trace browser. Moreover, there is on-going work to convert the traces to the Paraver format so that they can be analysed using Paraver (visual exploration) and Dimemas (what-if analysis).

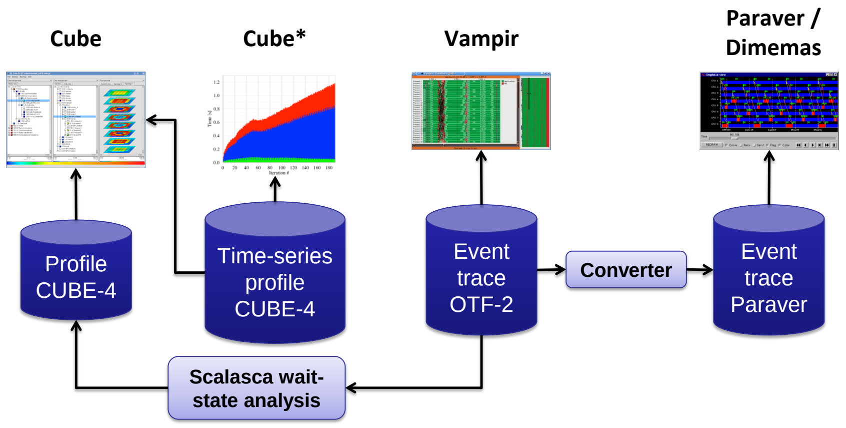


Fig. 3 Performance data types supported by Score-P and the tools that can be used to analyse them. The * next to the second mentioning of Cube indicates a display type that will be provided in a future version.

Overhead Minimisation

Another important aspect is the quality of the collected performance data in terms of intrusion and their size. To keep both intrusion and data size small, the Score-P measurement system offers a systematic approach of expanding the level of detail while at the same time narrowing the measurement focus:

1. Generate a summary profile with generous instrumentation while measuring the overhead. If the overhead is too large ($> 10\%$), reduce instrumentation, for example, through the application of filter lists. Measure overhead again and iterate until the overhead is satisfactory.
2. Generate a new summary profile with acceptable overhead. This provides an overview of the performance behaviour across the entire execution time and allows the identification of suspicious call paths and processes.
3. Generate a time-series profile, which provides a separate summary profile for every iteration of the time-step loop. This shows to which degree the performance behaviour changes as the simulation progresses and allows the identification of iterations that warrant deeper analysis. A semantic compression algorithm will ensure that the size of time-series profiles stays within reasonable limits.
4. For the identified iterations, generate event traces. Event traces provide the highest level of detail and offer a number of interesting analysis options including automatic wait-state analysis and visual exploration.

2.4 The HOPSA Performance Tools

This section introduces the various HOPSA performance tools.

2.4.1 Dimemas

Dimemas [12] is a performance prediction tool for message-passing programs. The Dimemas simulator reconstructs the time behaviour of a parallel application using as input an event trace that captures the time resource demands (CPU and network) of a parallel application. The target machine is modeled by a reduced set of key factors influencing the performance that model linear components like the point-to-point transfer time as well as non-linear factors like resources contention or synchronisation. Using a simple model, Dimemas allows performing parametric studies in a very short time frame. The supported target architecture is a cloud of parallel machines, each one with multiple nodes and multiples CPUs per node allowing the evaluation of a very wide range of alternatives, despite the most common environment is a computing cluster. Dimemas can generate as part of its output a Paraver trace file, enabling the user to conveniently examine the simulated run and understand the application behaviour.

Typical questions Dimemas helps to answer

- How would my application perform in a future system?
- Can increasing the network bandwidth improve the application performance?
- Would my application benefit from asynchronous communication?
- Is my application limited by the network or by serialisation and dependency chains in my code?
- What is the sensitivity of my application to different system parameters?
- What would be the impact of accelerating specific regions of my code?

2.4.2 Paraver

Paraver [1] is a very flexible data browser that is part of the CEPBA-Tools toolkit. Its analysis power is based on two main pillars. First, its trace format has no semantics; extending the tool to support new performance data or new programming models requires no changes to the visualiser – just capturing such data in a Paraver trace. The second pillar is that the metrics are not hardwired in the tool but can be programmed. To compute them, the tool offers a large set of time functions, a filter module, and a mechanism to combine two timelines. This approach allows displaying a huge number of metrics with the available data. To capture the expert's knowledge, any view or set of views can be saved as a Paraver configuration file. After that, re-computing the view with new data is as simple as loading the saved file. The tool has been demonstrated to be very useful for performance analysis studies, giving much more details about the application behaviour than most other performance tools.

Performance information in Paraver is presented with two main displays that provide qualitatively different types of information. The *timeline display* represents the behaviour of the application along time and processes, in a way that easily conveys to the user a general understanding of the application behaviour and simple identification of phases and patterns. The *statistics display* provides numerical analysis of the data that can be applied to any user-selected region, helping to draw conclusions on where and how to focus the optimisation effort. See Figures 4 and 5 for an example of Paraver's main displays.

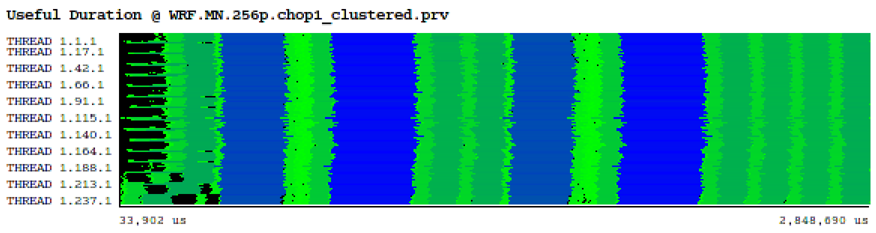


Fig. 4 Paraver timeline display.

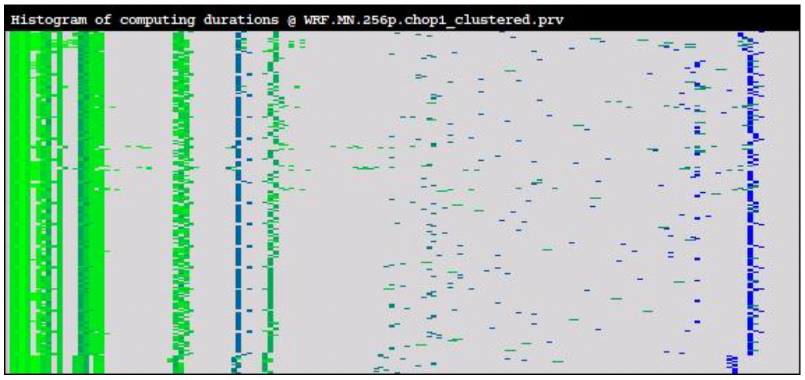

Histogram of computing durations @ WRF.MN.256p.chop1_clustered.prv

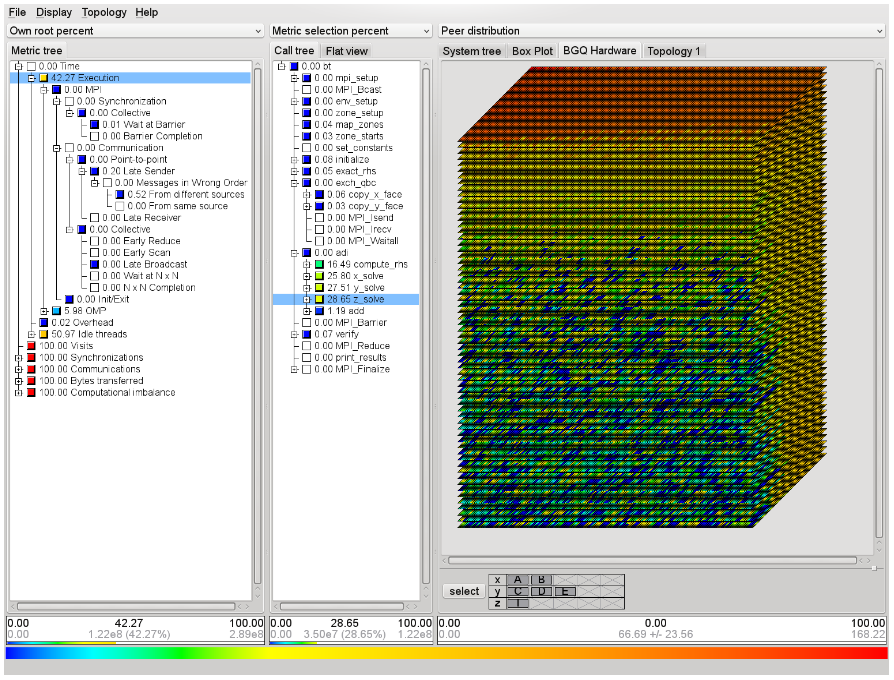
Fig. 5 Paraver histogram display.

Typical questions Paraver helps to answer

- What is the parallelisation efficiency and the performance of communication?
- What are the differences that can be observed between two different executions?
- Does the behaviour of the application change over time?
- Are performance or workload variations the cause of load imbalances in computation?
- Which performance issues do the microprocessor's hardware counters reflect?

2.4.3 Scalasca

Scalasca [2] is a free software tool that supports the performance optimisation of parallel programs by measuring and analysing their runtime behaviour. The tool has been specifically designed for use on large-scale systems including IBM Blue Gene and Cray XE, but is also well suited for small- and medium-scale HPC platforms. The analysis identifies potential performance bottlenecks – in particular those concerning communication and synchronization – and offers guidance in exploring their causes.



```
Own root percent
0.00 Time
  42.27 Execution
    0.00 MPI
      0.00 Synchronization
        0.00 Collective
          0.01 Wait at Barrier
          0.00 Barrier Completion
      0.00 Communication
        0.00 Point-to-point
          0.20 Late Sender
            0.00 Messages in Wrong Order
              0.52 From different sources
              0.00 From same source
          0.00 Late Receiver
        0.00 Collective
          0.00 Early Reduce
          0.00 Early Scan
          0.00 Late Broadcast
          0.00 Wait at N x N
          0.00 N x N Completion
      0.00 Init/Exit
    5.98 OMP
  0.02 Overhead
  50.97 Idle threads
100.00 Visits
100.00 Synchronizations
100.00 Communications
100.00 Bytes transferred
100.00 Computational imbalance
```

```
Metric selection percent
0.00 bt
  0.00 mpi_setup
  0.00 MPI_Bcast
  0.00 env_setup
  0.00 zone_setup
  0.04 map_zones
  0.03 zone_starts
  0.00 set_constants
  0.08 initialize
  0.05 exact_rhs
  0.00 exch_qbc
    0.06 copy_x_face
    0.03 copy_y_face
    0.00 MPI_Isend
    0.00 MPI_Irecv
    0.00 MPI_Waitall
  0.00 adi
    16.49 compute_rhs
    25.80 x_solve
    27.51 y_solve
    28.65 z_solve
    1.19 add
  0.00 MPI_Barrier
  0.07 verify
  0.00 MPI_Reduce
  0.00 print_results
  0.00 MPI_Finalize
```

```
Peer distribution
```

```
0.00      42.27              100.00 | 0.00      28.65              100.00 | 0.00          0.00             100.00
0.00      1.22e8 (42.27%)    2.89e8 | 0.00      3.50e7 (28.65%)    1.22e8 | 0.00          66.69 +/- 23.56  168.22
```

Fig. 6 Interactive exploration of performance behaviour in Scalasca along the dimensions performance metric (left), call tree (middle), and process topology (right). Screendump shows result of a 524,288 threads run on the Jülich BlueGene/Q machine.

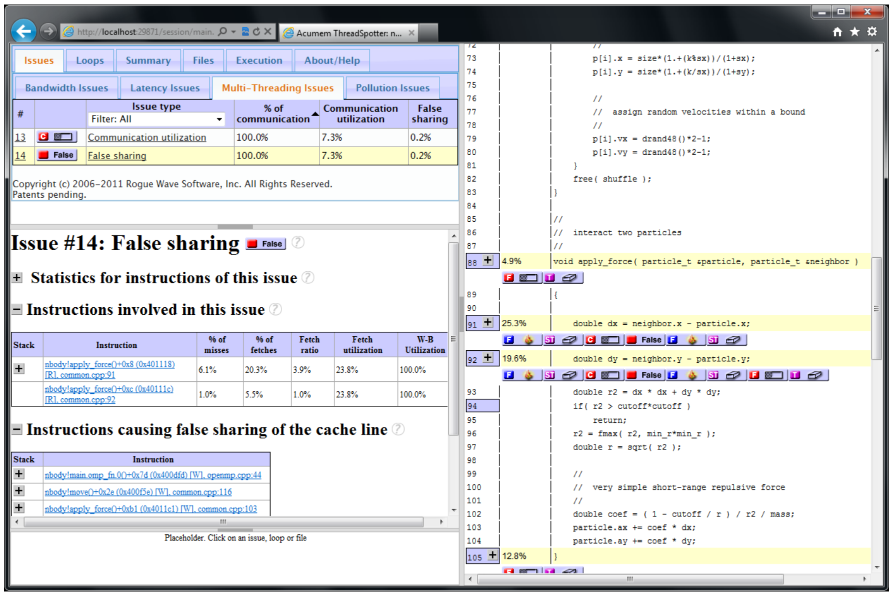
The user of Scalasca can choose between two different analysis modes: (i) performance overview on the call-path level via profiling and (ii) the analysis of wait-state formation via event tracing. Wait states often occur in the wake of load imbalance and are serious obstacles to achieving satisfactory performance. Performance-analysis results are presented to the user in an interactive explorer called Cube (Figure 6) that allows the investigation of the performance behaviour on different levels of granularity along the dimensions performance problem, call path, and process. The software has been installed at numerous sites in the world and has been successfully used to optimise academic and industrial simulation codes.

Typical questions Scalasca helps to answer

- Which call-paths in my program consume most of the time?
- Why is the time spent in communication or synchronisation higher than expected?
- Does my program suffer from load imbalance and why?

2.4.4 ThreadSpotter

ThreadSpotter [3] is a commercial tool that will help programmers optimise their programs with respect to architectural bottlenecks such as cache size and memory system bandwidth and point out inefficient communication modes between threads. Its scope is a single process, including both single-threaded as well as multi-threaded applications.



#		Issue type Filter: All	% of communication	Communication utilization	False sharing
13	C	Communication utilization	100.0%	7.3%	0.2%
14	False	False sharing	100.0%	7.3%	0.2%

Copyright (c) 2006–2011 Rogue Wave Software, Inc. All Rights Reserved.
Patents pending.

Issue #14: False sharing False

+ Statistics for instructions of this issue

− Instructions involved in this issue

Stack	Instruction	% of misses	% of fetches	Fetch ratio	Fetch utilization	W-B Utilization
+	nbody!apply_force()+0x8 (0x401118) [R], common.cpp:91	6.1%	20.3%	3.9%	23.8%	100.0%
	nbody!apply_force()+0xc (0x40111c) [R], common.cpp:92	1.0%	5.5%	1.0%	23.8%	100.0%

− Instructions causing false sharing of the cache line

Stack	Instruction
+	nbody!main.omp_fn.0()+0x7d (0x400dfd) [W], openmp.cpp:44
+	nbody!move()+0x2e (0x400f5e) [W], common.cpp:116
+	nbody!apply_force()+0xb1 (0x4011c1) [W], common.cpp:103

Placeholder. Click on an issue, loop or file

```
                //
                p[i].x = size*(1.+(k%sx))/(1+sx);
                p[i].y = size*(1.+(k/sx))/(1+sy);

                //
                //  assign random velocities within a bound
                //
                p[i].vx = drand48()*2-1;
                p[i].vy = drand48()*2-1;
            }
            free( shuffle );
        }

        //
        //  interact two particles
        //
4.9%    void apply_force( particle_t &particle, particle_t &neighbor )
        {

25.3%       double dx = neighbor.x - particle.x;
19.6%       double dy = neighbor.y - particle.y;
            double r2 = dx * dx + dy * dy;
            if( r2 > cutoff*cutoff )
                return;
            r2 = fmax( r2, min_r*min_r );
            double r = sqrt( r2 );

            //
            //  very simple short-range repulsive force
            //
            double coef = ( 1 - cutoff / r ) / r2 / mass;
            particle.ax += coef * dx;
            particle.ay += coef * dy;
12.8%   }
```

Fig. 7 Highlighting a ”false sharing” situation. Top left part contains lists of problems. Lower left contains details, and annotated source code is to the right.

Some programming styles will exercise the memory system in suboptimal ways that can reduce performance drastically. Examples of these are failure to observe or exploit locality properties in code or data. Inappropriate communication through shared memory between threads may cause the coherence traffic to become a bottleneck.

ThreadSpotter explains the inefficiencies of observed memory access patterns on a high level in a graphical user interface (Figure 7) and provides pointers to suggestions to optimise the code. It offers deep explanations on hardware level to back up the suggestions, educating the user as he uses the tool.

Typical questions ThreadSpotter helps to answer

- How does my program abuse the memory system and what can I do about it?
- Do the threads of my program exchange data with each other in an inefficient way?
- When adjusting my program, are the changes actually helping to minimise the footprint of the application?

2.4.5 Vampir

Vampir [4] is a graphical analysis framework that provides a large set of different chart representations of event-based performance data. These graphical displays, including timelines and statistics, can be used by developers to obtain a better understanding of their parallel program's inner working and to subsequently optimise it. See Figure 8 for an impression of the Vampir GUI.

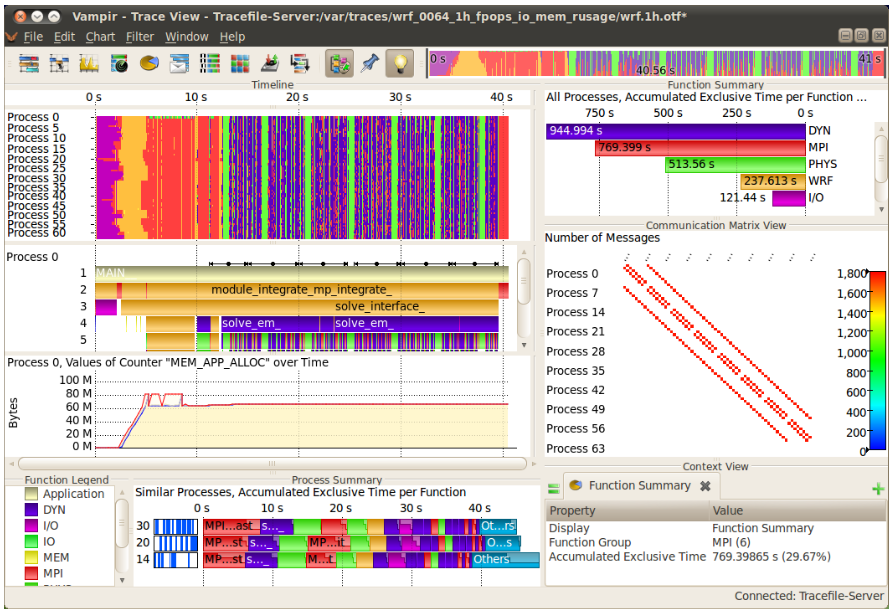

Vampir - Trace View - Tracefile-Server:/var/traces/wrf_0064_1h_fpops_io_mem_rusage/wrf.1h.otf*
File Edit Chart Filter Window Help
Timeline
Function Summary
All Processes, Accumulated Exclusive Time per Function ...
944.994 s DYN
769.399 s MPI
513.56 s PHYS
237.613 s WRF
121.44 s I/O
Communication Matrix View
Number of Messages
module_integrate_mp_integrate_
solve_interface_
solve_em_
Process 0, Values of Counter "MEM_APP_ALLOC" over Time
Function Legend
Application
DYN
I/O
IO
MEM
MPI
Process Summary
Similar Processes, Accumulated Exclusive Time per Function
Context View
Function Summary
Property Value
Display Function Summary
Function Group MPI (6)
Accumulated Exclusive Time 769.39865 s (29.67%)
Connected: Tracefile-Server

Fig. 8 Vampir GUI

Vampir is designed to be an intuitive tool, with a GUI that enables developers to quickly display program behavior at any level of detail. Different timeline displays show application activities and communication along a time axis, which can be zoomed and scrolled. Statistical displays provide quantitative results for the currently selected time interval. Powerful zooming and scrolling along the timeline and process/thread axis allows pinpointing the causes of performance problems. All displays have context-sensitive menus, which provide additional information and customisation options. Extensive filtering capabilities for processes, functions, messages or collective operations help to narrow down the information to the interesting spots. Vampir is based on Qt and is available for all major workstation operation systems as well as on most parallel production systems. The parallel version of Vampir, VampirServer, provides fast interactive analysis of ultra large data volumes.

Typical questions Vampir helps to answer

- What happens in my application execution during a given time in a given process or thread?
- How do the communication patterns of my application execute on a real system?
- Are there any imbalances in computation, I/O or memory usage and how do they affect the parallel execution of my application?

2.5 Integration among Performance Analysis Tools

Sharing the common measurement infrastructure Score-P and its data formats and providing conversion utilities if direct sharing is not possible, the performance tools in the HOPSA environment and workflow already make it easier to switch from higher-level analyses provided by tools like Sclasca to more in-depth analyses provided by tools like Paraver or Vampir. To simplify this transition even further, the HOPSA tools are integrated in various ways. With its automatic trace analysis, Scalasca locates call paths affected by wait states caused by load or communication imbalance. However, to find and fix these problems in a user application, it is in some cases necessary to understand the spatial and temporal context leading to the inefficiency, a step naturally supported by trace visualizers like Paraver or Vampir. To make this step easier, the Scalasca analysis remembers the worst instance for each of the performance problems it recognizes. Then, the Cube result browser can launch a trace browser and zoom the timeline into the interval of the trace that corresponds to the worst instance of the recognized performance problems.

In the future, the same mechanisms will be available for a more detailed visual exploration of the results of Scalasca's root cause analysis as well as for further analyzing call paths involving user functions that take too much execution time. For the latter, ThreadSpotter will be available to investigate their memory, cache and multi-threading behaviour. If a ThreadSpotter report is available for the same

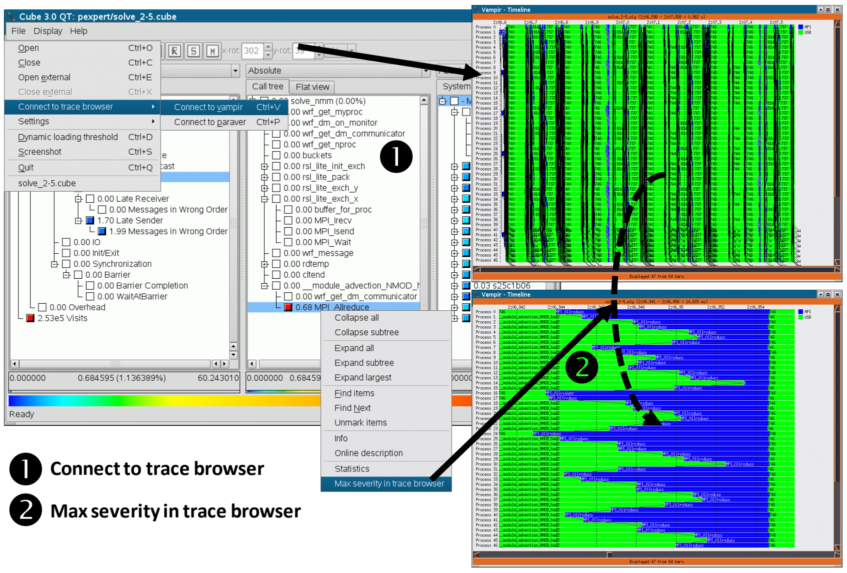


Fig. 9 Scalasca to Vampir or Paraver Trace browser integration. In a first step, when the user requests to connect to a trace browser, the selected visualizer is automatically started and the event trace, which was previously the basis of Scalasca's trace analysis, is loaded. Now, in a second step, the user can request a timeline view of the worst instance of each performance bottleneck identified by Scalasca. The trace browser view automatically zooms to the right time interval. Now the user can use the full analysis power of these tools to investigate the context of the identified performance problem.

executable and dataset, Cube will allow launching detailed ThreadSpotter views for each call path where data from both tools is available.

2.6 *Integration of System Data and Performance Analysis Tools*

The Russian ClustrX.Watch management software provides node-level sensor information that can give additional insight for performance analysis of applications with respect to the specific system they are running on. This allows populating Paraver and Vampir traces with system information (the granularity will depend on the overhead to obtain the data) and to analyze them with respect to the system-wide performance.

The Russian LAPTA system data analysis and management software provides node-level sensor information that can give additional insight for performance analysis of applications with respect to the specific system they are running on. This allows populating Paraver and Vampir traces with system information collected by Clustrx, Ganglia, and other sources (the granularity will depend on the overhead to

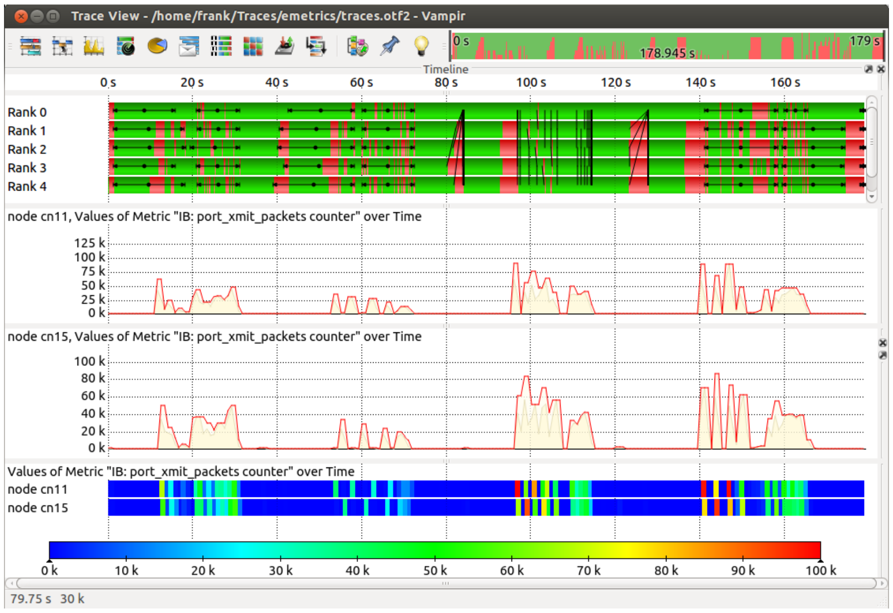

Trace View - /home/frank/Traces/emetrics/traces.otf2 - Vampir
0 s
178.945 s
179 s
Timeline
0 s
20 s
40 s
60 s
80 s
100 s
120 s
140 s
160 s
Rank 0
Rank 1
Rank 2
Rank 3
Rank 4
node cn11, Values of Metric "IB: port_xmit_packets counter" over Time
125 k
100 k
75 k
50 k
25 k
0 k
node cn15, Values of Metric "IB: port_xmit_packets counter" over Time
100 k
80 k
60 k
40 k
20 k
0 k
Values of Metric "IB: port_xmit_packets counter" over Time
node cn11
node cn15
0 k
10 k
20 k
30 k
40 k
50 k
60 k
70 k
80 k
90 k
100 k
79.75 s 30 k

Fig. 10 Vampir screendump showing aligned system and application data.

obtain the data) and to analyze them with respect to the system-wide performance. The system offers two different ways to access to the collected data:

Historic information is stored with a given granularity for all the sensors and all the IP (nodes) on the system. The initial granularity was very coarse (one minute) and did not seem useful for the population of application trace files because there can be many different program phases in a one minute interval. On the other hand, the circular buffer provides historical information with fine-grained detail (coarser or equal to 1 second depending on the sensor) for the last minutes (300 measurements). *Streamed information* can be requested for any range of sensors and IPs. The interface provides at least a value every 10 seconds unless there is a change greater than a 10%. Currently, the finest available granularity is 1 second.

Both mechanisms use a connection through an HTTP protocol that in the case of the streamed data has to be refreshed periodically or dies after 5 minutes. We evaluated both alternatives to see their potential and identified possible drawbacks.

The Vampir team implemented a prototype Score-P adapter that enhances OTF2 traces at the end of the measurement. For evaluation, the benchmark code HPL was instrumented with Score-P. In addition to the application and MPI events, the trace was enhanced with HOPSA node-level metrics and per-process PAPI counters. Tested and working HOPSA sensors include node memory usage values and Infiniband packet counts. The evaluation shows that phases in the application clearly correlate to measured values of the node level sensors, e.g. heavy MPI communication to Infiniband packet counters (see Figure 10. As a use case, this integration allows the user to analyze how the application utilizes network hardware of each node or how shared usage of network resources affects the application execution. Currently the sensor values are available in 1 second granularity for the last 500 seconds and 1 minute granularity before that.

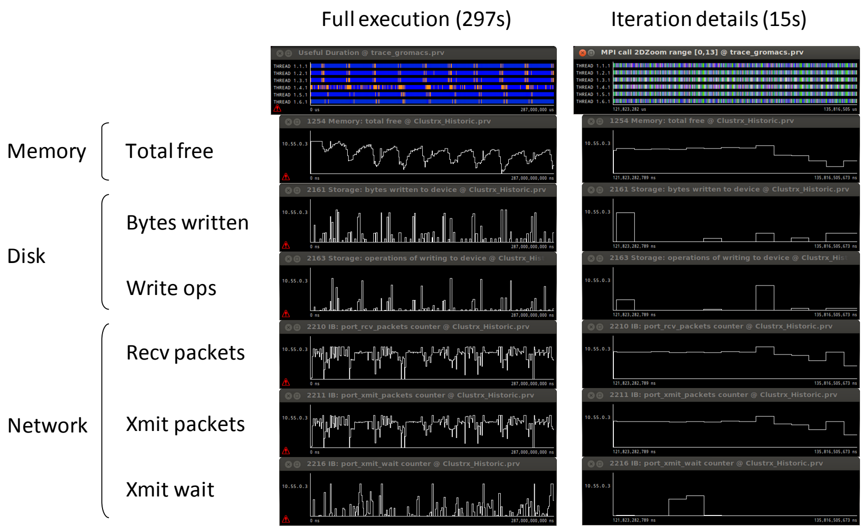


Fig. 11 Paraver screendumps showing aligned system and application data.

The Paraver team experimented with Gromacs, a popular production code in life sciences, trying to correlate the sensors values to the activity of the application As one can see in Figure 11, on a higher level, it is possible to correlate system metrics with application program phases (left side of the picture). However, due to the limited resolution of the system metrics data, this is not possible on a more detailed level (see right side of the figure).

2.7 *Opportunities for System Tuning*

Several opportunities for system tuning arise from the availability of historic performance data collected by LWM^2 . First, data on individual system nodes along an extended period of time in comparison to other nodes can be analysed to spot anomalies and detect deficient components. Second, data on the entire workload can be used to improve the understanding of the workload requirements and configure the system accordingly. The insights obtained may guide the evolution of the system and influence future procurement decision. Finally, knowledge of the resource requirements of individual jobs offers the chance to develop resource-aware scheduling algorithms that avoid oversubscription of shared resources such as the file system or the network.

3 Conclusions

The HOPSA project creates an integrated diagnostic infrastructure for combined application and system tuning. Starting from system-wide basic performance screening of individual jobs, an automated workflow routes findings on potential bottlenecks either to application developers or system administrators with recommendations on how to identify their root cause using more powerful diagnostics. This document specifies the performance analysis workflow that connects the different steps. At the same time, it provides an impression of the overall vision behind the project. The high- level description is intended to make it readable also for non-tool experts.

Although the specification is based on long experience with HPC application developers and how they tend to use performance tools, it is a blueprint that needs to be validated in practice. This validation is planned for the last quarter of the project at Moscow State University, once all the components are in place and, in particular, LWM^2 has been fully completed, tested, and integrated into the overall environment. During this validation process, some of the details presented in this document may change and ultimately result in a new revision. We expect though that all major elements will be retained.

Beyond the lifetime of the project, the HOPSA infrastructure is supposed to collect large amounts of valuable data on the performance of individual applications as well as the system workload as a whole. It will be of interest in three ways: to tune individual applications, to tune the system for a given workload, and finally to observe the evolution of this workload over time. The latter will allow the effectiveness of our strategy to be studied. An open research issue to be tackled on the way will be the reliable tracking of individual applications, which may change over time, across jobs based on the collected data. In this way, it will become possible to document the performance history of code projects and demonstrate the effects of our tool environment over time.

Acknowledgements HOPSA is a coordinated twin project funded under FP7-ICT-2011-EU-Russia grant number FP7-277463 and Russian Ministry of Education and Science contract number 07.514.12.4001. The authors also would like to thank our collegues working with us on this project: Andrew Adinetz, Daniel Becker, Peter Bryzgalov, Jens Domke, Markus Geimer, Juan Gonzalez, André Grötzsch, Thomas Ilsche, Germán Llort, Christopher Schleiden, Konstantin Stefanov, Zoltán Szebenyi, Igor Zacharov, Pavel Saviankou, Igor Ustinov, Vadim Voevodin, and Sergey Zhumatiy as well as the Paraver, Scalasca, and Vampir teams in general.

References

1. J. Labarta, S. Girona, V. Pillet, T. Cortes, L. Gregoris, DiP: A parallel program development environment. in: Proc. of the 2nd International Euro- Par Conference, Lyon, France, Springer, 1996.
2. M. Geimer, F. Wolf, B.J.N. Wylie, E. brahm, D. Becker, B. Mohr: The Scalasca performance toolset architecture. Concurrency and Computation: Practice and Experience, 22(6):702–719, April 2010.
3. E. Berg, E. Hagersten: StatCache: A Probabilistic Approach to Efficient and Accurate Data Locality Analysis. In: Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2004), Austin, Texas, USA, March 2004.
4. W. Nagel, M. Weber, H.-C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and Analysis of MPI Resources. Supercomputer, 12(1):69–80, 1996.
5. D. an Mey, S. Biersdorff, C. Bischof, K. Diethelm, D. Eschweiler, M. Gerndt, A. Knüpfer, D. Lorenz, A.D. Malony, W.E. Nagel, Y. Oleynik, C. Rössel, P. Saviankou, D. Schmidl, S.S. Shende, M. Wagner, B. Wesarg, F. Wolf: Score-P: A Unified Performance Measurement System for Petascale Applications. In: Competence in High Performance Computing 2010 (CiHPC), pp. 85–97. Gauß-Allianz, Springer (2012).
6. M. Gerndt and M. Ott. Automatic Performance Analysis with Periscope. Concurrency and Computation: Practice and Experience, 22(6):736–748, 2010.
7. S. Shende and A. D. Malony. The TAU Parallel Performance System. International Journal of High Performance Computing Applications, 20(2):287–331, 2006. SAGE Publications.
8. M. Geimer, P. Saviankou, A. Strube, Z. Szebenyi, F. Wolf, B. J. N. Wylie: Further improving the scalability of the Scalasca toolset. In: Proceedings of PARA 2010: State of the Art in Scientific and Parallel Computing, Part II: Minisymposium Scalable tools for High Performance Computing, Reykjavik, Iceland, June 6–9 2010, volume 7134 of Lecture Notes in Computer Science, pages 463–474, Springer, 2012.
9. D. Eschweiler, M. Wagner, M. Geimer, A. Knpfer, W. E. Nagel, F. Wolf: Open Trace Format 2 - The Next Generation of Scalable Trace Formats and Support Libraries. In: Proceedings of the International Conference on Parallel Computing (ParCo), Ghent, Belgium, 2011, volume 22 of Advances in Parallel Computing, pages 481–490, IOS Press, 2012.
10. A. Knüpfer, R. Brendel, H. Brunst, H. Mix, W. E. Nagel: Introducing the Open Trace Format (OTF), In: Vassil N. Alexandrov, Geert Dick van Albada, Peter M. A. Sloot, Jack Dongarra (Eds): Computational Science - ICCS 2006: 6th International Conference, Reading, UK, May 28–31, 2006, Proceedings, Part II, Springer Verlag, ISBN: 3-540-34381-4, pages 526–533, Vol. 3992, 2006.
11. F. Wolf, B. Mohr: EPILOG Binary Trace-Data Format. Technical Report FZJ-ZAM-IB-2004-06, Forschungszentrum Jülich, 2004.
12. H. Servat Gelabert, G. Llort Sanchez, J. Gimenez, and J. Labarta. Detailed performance analysis using coarse grain sampling. In: Euro-Par 2009 - Parallel Processing Workshops, Delft, The Netherlands, August 2009, volume 6043 of Lecture Notes in Computer Science, pages 185–198. Springer, 2010.
13. T-Platforms, Moscow, Russia. Clustrx HPC Software. `http://www.t-platforms.com/products/software/clustrxproductfamily.html`, last accessed September 2012.
14. A.V. Adinets, P.A. Bryzgalov, Vad.V. Voevodin, S.A. Zhumatiy, D.A. Nikitenko. About one approach to monitoring, analysis and visualization of jobs on cluster system (In Russian). In: Numerical Methods and Programming, 2011, vol. 12, Pp. 90–93
15. Z. Szebenyi, F. Wolf, B. J.N. Wylie. Space-Efficient Time-Series Call-Path Profiling of Parallel Applications. In: Proc. of the ACM/IEEE Conference on Supercomputing (SC09), Portland, OR, USA, ACM, 2009.