

# IMPuLSE: Integrated Monitoring and Profiling for Large-Scale Environments\*

Patrick G. Bridges  
Department of Computer Science  
MSC01-1130  
1 The University of New Mexico  
Albuquerque, New Mexico 87131  
bridges@cs.unm.edu

Arthur B. Maccabe  
Department of Computer Science  
MSC01-1130  
1 The University of New Mexico  
Albuquerque, New Mexico 87131  
maccabe@cs.unm.edu

## ABSTRACT

A lack of efficient system software is an increasing impediment to deploying large-scale parallel and distributed systems. Systemically addressing operating system-induced performance anomalies requires accurate, low-overhead, whole-system monitoring, something that is currently unavailable in large tightly-coupled systems. In this paper, we present the design of IMPuLSE—Integrated Monitoring and Profiling for Large-Scale Environments—a system we are developing to meet this need. IMPuLSE’s innovative message-centric profiling approach trades off of centralized global knowledge for low overhead, while retaining relatively fine-grained information about important cross-host performance interactions. The goal of this approach is to enable both large-scale system software adaptation and continuous system performance auditing.

## 1. INTRODUCTION

System software performance is an increasing limitation on application scalability on both current and emerging large-scale systems. Recent research in large-scale cluster environments has shown that even seemingly insignificant operating system interference can result in large application slowdowns [17]. Other research has shown that operating system structure and scheduling decisions can have performance large effects in large-scale systems [5, 10]. These problems will be even more significant in emerging systems with tens or hundreds of thousands compute nodes.

A variety of powerful techniques exist for addressing system software performance issues, including application-specific operating system and protocol configuration [4, 7] and system software adaptation [3, 16]. These techniques, however, require lightweight online methods for measuring system performance. No adequate techniques exist for such monitoring in large-scale parallel systems, leading to the myriad system software problems in modern large-

\*This work was supported in part under subcontract R7A824-7920004 from the Los Alamos Computer Science Institute and Rice University.

scale environments.

Innovative techniques for lightweight online monitoring in tightly-integrated, large-scale environments are needed to enable a wide range of new system software research. Such techniques could be used by anomaly detection systems that are currently feasible only in loosely-coupled distributed systems [2] and to design new scalable operating and runtime system services for large-scale systems. These techniques would also enable new studies of application needs which could, in turn, lead to in operating systems, programming models, and system services that can be adapted according to application needs.

In this paper, we describe the design of IMPuLSE—Integrated Monitoring and Profiling for Large-Scale Environments—a profiling system we are developing to provide lightweight system-wide performance monitoring in large-scale parallel and distributed systems. IMPuLSE is designed to collect performance data across multiple software layers, including the operating system, middleware, and applications, and to propagate this information across the system to enable both online and offline performance analysis and adaptation. IMPuLSE’s design is unique in that it takes a message-centric as opposed to host-centric view of performance monitoring, allowing it to combine cross-system performance information without requiring a centralized data collection point. By combining this approach with other novel techniques, IMPuLSE strives to reduce monitoring overhead to near-imperceptible levels.

## 2. PRIOR WORK

A great deal of prior work exists on performance monitoring in parallel and distributed systems. Trace-based approaches such as Paradyn [14], VAMPIR [15], and Netlogger [19] collect detailed data about a wide range of events and can correlate these events across machines to obtain a global view of system behavior. Some recent work [1, 2, 6] has also sought to use traces for per-request post-mortem performance and fault analysis in loosely-coupled distributed systems. Unfortunately, trace-based approaches normally have large overheads in tightly-coupled parallel systems that can perturb system behavior, as the generated traces must be saved to disk or sent over the network while the system is running. In addition, traces can be difficult to use for online adaptation because they are generally combined post-mortem.

Statistical monitoring systems[9, 12, 13, 18] represent a much lower-overhead approach based on hardware performance counters. While such information is available online and can be used to make adap-

tation decisions, it is usually comparatively coarse-grained and difficult to relate to specific causes. This coarse granularity also makes it difficult to correlate or combine such data across nodes to determine cross-host effects.

Other recent work attempts to utilize existing message traffic to gather more detailed information about interactions between nodes. PHOTON, for example, places a small amount of data in each outgoing MPI message to identify the message context and when it was sent [20]. The receiver can then use this information to keep track of information such as average message latencies without introducing round-trip ping-pong traffic. PHOTON does not, however, seek to gather more comprehensive performance statistics on operating system performance and does not attempt to correlate and combine performance information from multiple messages. The Wren network-monitoring system [22] takes a similar passive-monitoring approach, using existing messages to monitor network performance for grid-based systems. Neither of these systems use existing messages to propagate performance between nodes in a running system.

### 3. SYSTEM DESIGN

IMPuLSE is designed to achieve a balance between the fine-grained global view available in trace-based systems and the low overhead, online availability of statistical approaches. It does this by trading away the centralized global view provided by tracing for a decentralized view. To achieve this, IMPuLSE’s design includes a number of novel techniques:

- A message-centric approach to profiling that associates profiling data with incoming and outgoing messages instead of individual hosts.
- A two-dimensional design where vertical profiling covers profiling across system layers within a host and horizontal profiling propagates and combines performance data from multiple hosts.
- A system-wide statistical sampling strategy to reduce profiling overhead while still allowing the propagation of profiling information across the system.

#### 3.1 Message-Centric Profiling

With the exception of some recent work [1, 2, 6, 20], existing parallel and distributed performance monitoring systems are host-centric—they collect and analyze data on a host-by-host basis. The increasing speed of modern processors, however, means that parallel and distributed systems are generally performance-limited by communication costs. To capture this behavior, IMPuLSE takes a *message-centric* approach to performance monitoring that associates performance data with the messages that traverse through the system. The goal of message-centric profiling is to have every message include a summary of important performance metrics on the critical path of generating and communicating the data contained in the message. This information should encompass costs such as network latency, scheduling overhead, and application processing.

Figure 1 illustrates how performance data is exchanged in IMPuLSE’s message-centric approach. When node 1 sends a new message that is unrelated to other messages, it places a summary of the CPU time required to generate the message in the message’s *profiling header*. As this message is sent by one host and received by another, the

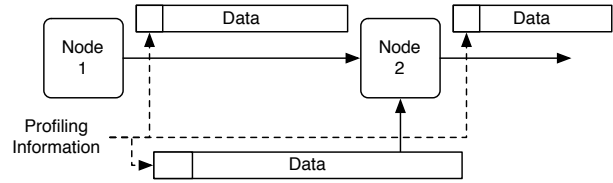


Figure 1: Message-centric Profiling Example

system software layers augment the message with additional performance data on the amount of time required to send and receive the message (overhead). The receiving host then combines this performance data with performance data from other related messages and the local host. This new performance summary is then included in outgoing messages.

Representing performance monitoring data in message-centric monitoring is a fundamental tradeoff between accuracy and overhead. IMPuLSE’s design currently uses a 92-byte profiling header in each message to contain performance summaries; we believe that this will keep overheads low while allowing the system to monitor key message-passing costs. This header consists of several fields, with the majority of the space dedicated to nine 64-bit counters, each of which contains the amount of time spent on a particular task. A separate portion of the header contains eight 8-bit tags that describe which task each of eight of the counters is monitoring. The ninth counter is designated as the `OTHER` counter, and contains the amount of time spent on tasks not explicitly counted in the other seven counters. Each time the profiling header is modified, the performance metrics with the highest total times are placed in the specific counter fields, with other times folded into the `OTHER` counter. This process is obviously lossy, but necessary to limit overhead in a message-centric profiling systems.

The profiling header also includes a hop count (32 bits) that allows IMPuLSE to control the distance that performance data can propagate in a system and a counter selection bit-field (64 bits) that IMPuLSE uses to disable the inclusion certain performance metrics in the explicit counter fields. These two fields are used for system-wide statistical sampling in IMPuLSE to overcome some of the lossy compression of performance data, as described in section 3.3

#### 3.2 Two-dimensional Monitoring

IMPuLSE’s design is decomposed into two complementary dimensions:

**Vertical profiling** is responsible for collecting performance data inside a single host as a message traverses its software layers.

**Horizontal profiling** is responsible for combining performance data from multiple messages prior to the transmission of new outgoing messages so that this data propagates between hosts.

##### 3.2.1 Vertical Profiling

Vertical profiling implementations are by nature system-dependent, and most commodity operating systems do not include systematic ways to gather performance information on a per-request or per-packet basis. We are currently prototyping IMPuLSE on Linux, where we are using a modified version of the Linux Tracing Toolkit

[21] to collect vertical profiling information. IMPuLSE currently uses a simple linear fit for predicting network latencies. In cases where network latencies they are not statically predictable, however, existing network performance monitoring techniques such as Wren [22] will be used.

### 3.2.2 Horizontal Profiling

In contrast to vertical profiling, horizontal profiling in IMPuLSE is largely machine independent; its interface to the vertical profiling system is used to retrieve and place profiling information in messages. Horizontal profiling is responsible for taking the performance data collected on other machines (through both horizontal and vertical profiling) and combining this with the data recorded on message reception. The resulting performance summary is then merged into outgoing messages.

The primary challenge for horizontal profiling is determining transitivity between received performance information and local performance information. IMPuLSE's current approach to addressing this problem is to determine whether received performance data *dominates* locally stored performance information on every message reception. If received performance data dominates local performance information, the received information supplants local information; if the local information dominates, on the other hand, received performance information is discarded. Determining domination relationships between performance summaries is sufficient to for monitoring the critical path in parallel and distributed applications; its main shortcoming is that it loses information from off-critical paths that may limit the effectiveness of critical-path-based optimizations.

IMPuLSE determines domination of performance data in MPI applications by monitoring calls to `MPI_wait` and blocking sends and receives. If the application blocks waiting for incoming data or acknowledgments, the performance data in the waited-for message dominates locally gathered monitoring information. If, on the other hand, the application uses non-blocking sends and receives and never waits for data arrival (*i.e.*, `MPI_wait` returns immediately), the local performance data dominates the received monitoring summary, which is discarded.

### 3.3 System-wide Statistical Sampling

Because our proposed data representation involves the lossy compression of multiple data samples in an effort to limit monitoring overhead (the `OTHER` counter), important performance details may be lost. In systems with many sources of overhead of approximately equal size, the top overheads may prevent the collection of detailed information on other important overheads. This is particularly troubling when the largest overheads are not removable.

IMPuLSE uses the counter selector bit-field and hop count field in the profiling header to address this problem. In cases where monitoring overhead is excessive or performance data is not detailed enough, two machines on opposite sides of the system will be designated as the only ones to introduce new performance data. Intermediary machines will only propagate received performance information and augment it with their local data. In addition, the hop-count in newly-introduced performance data will be set to approximately the width of the computation in nodes.

The two designated machines will alternate introducing messages with performance data into the system; after the initial performance measurement, a designated machines will introduce a new perfor-

mance message exchange only after removing the hop-count limited message introduced by its partner machine. Once a monitoring host determines that the performance data has reached a steady-state, it will disable inclusion of the top performance metrics in the new message and introduce a new message exchange to measure lower-overhead effects that may, for example, be more amenable to adaptation.

Because this technique allows sampling to be done infrequently, such a sample-based approach to message-centric profiling should reduce the average overhead of monitoring to nearly imperceptible levels. This would enable its inclusion in production applications. Such audits could then be compared with known performance profiles or analyzed by data mining software to detect performance anomalies resulting from new data sets or system software changes. In addition, it would also make it easier to build adaptive high-performance applications by reducing or eliminating the overheads that can prevent adaptations from reaching break-even performance levels.

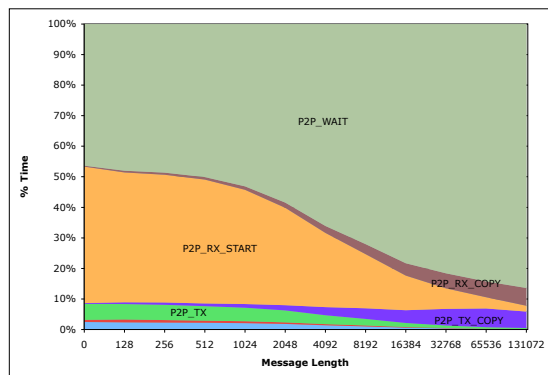
## 4. PRELIMINARY RESULTS

To demonstrate the viability of IMPuLSE's approach, we have implemented a simple message-centric profiling prototype for point-to-point messages in the Los Alamos MPI implementation (LA-MPI) [8]. This implementation uses the built-in Pentium III cycle counters to measure the amount of time spent in various parts of the LA-MPI implementation, estimates message-passing latency between machines using a piecewise linear fit to low-level message passing latencies, and propagates performance statistics to remote nodes in MPI message headers and acknowledgments. Monitoring summaries in this prototype are 96 bytes long and include 8 specific counters, the `OTHER` counter, an extra counter specifically for measuring monitoring overheads, but no 4-byte hop-count.

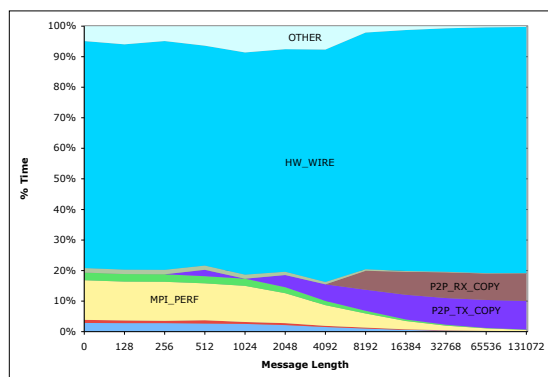
As we are initially interested in quantifying added message-passing overhead, our initial tests have used a simple MPI ping-pong latency and bandwidth test. The performance monitoring system measures the total amount of time to produce 250,000 MPI round trips of a configurable size between two 2 GHz Pentium III Xeon machines connected by Myrinet Lanai V cards.

Figure 2 shows the breakdown of where the performance monitoring system measures where time is spent both with host-only monitoring and message-centric monitoring. By propagating performance information across the system in messages, message-centric profiling quantifies where time is actually spent during the complete message exchange, in this case predominantly on the wire and, for larger messages, copying data. Host-based monitoring, in contrast, can only determine that the majority of the time is spent waiting (`P2P_WAIT`) and checking if new messages have arrived (`P2P_RX_START`). This preliminary work demonstrates that message-centric monitoring is better able to provide a direct indication how a system could be modified to improve application performance, in this case by removing copies (`P2P_TX_COPY` and `P2P_RX_COPY`) and using a lower-latency network interface.

The monitoring system measures its own overhead as approximately 13% for small messages, dropping rapidly as message size increases. Extra MPI latency is approximately 7.2 microseconds per round trip. A recent study by a UNM student [11] estimates that an added 3.6 microseconds of per-message overhead will cause an average slowdown of approximately 9% in a variety of applications from the ASCI Purple and NAS B parallel benchmark suites. Note that



(a) Host-centric Monitoring



(b) Message-centric Monitoring

Figure 2: Ping-pong time breakdown measured using host-centric and message-centric monitoring

this overhead would also be significantly reduced on 64-bit machines, as then native 64-bit math could be used directly for counter arithmetic. We have not yet directly measured application overheads because of implementation limitations in the current prototype.

## 5. REMAINING CHALLENGES

While we have finalized the majority of the design decisions for IMPuLSE and are in the process of implementation, a number of research questions remain unresolved, particularly in relation to horizontal profiling. As previously mentioned, IMPuLSE uses a simple dominating relationship to determine how to propagate received performance data. While this assumption is sufficient to characterize the critical path in parallel and distributed applications, it does not take into account how far that critical path is from other longer computational paths in the system. For example, the critical path may show that protocol overheads constitute 25% of the critical path time because remotely received data almost always dominates local computation. This can happen when local computation always finishes just before message reception. In this case, removing

protocol overheads from the critical path would not greatly speed up program performance. We fully expect to explore more sophisticated approaches to merging local and remotely received performance data to try and better characterize which parts of the system critical path are most amenable to adaptation.

Another unresolved issue in the current design of IMPuLSE is how to account of for secondary performance effects. As an example, the delivery of a message B may be delayed waiting for the delivery of message A, which is itself waiting for the target process to be scheduled. In IMPuLSE's current design, the delay in delivering message A would correctly be accounted for, but the delay in delivering message B would be attributed to message queuing delays. Ideally, IMPuLSE should be able to detect some second order effects and account for them properly. The performance anomalies that have recently been discovered in large-scale machines are primarily due to such effects, so detecting them and accounting for them correctly could be a great aid in making adaptation decisions and in performance debugging and tuning.

## 6. CONCLUSIONS

IMPuLSE's approach to large-scale system performance monitoring represents a novel trade-off between global knowledge, accuracy, and overhead compared compared to existing systems. By sacrificing complete global knowledge, IMPuLSE aims to achieve the low monitoring overheads of existing host-centric monitoring systems based solely on hardware counters while retaining the ability to measure the most important cross-host performance effects at a relatively fine granularity. In addition, IMPuLSE's message-centric approach is designed to allow it to measure the performance of both application software and system software, naturally propagating performance information through a running system. When combined with appropriate statistical sampling techniques, we believe IMPuLSE will provide the low-overhead, system-wide monitoring necessary to assure reliable performance in emerging large-scale machines and to enable innovative system-wide adaptation techniques.

## 7. REFERENCES

- [1] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP-19)*, Bolton Landing, NY, 2003.
- [2] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan. Magpie: online modelling and performance-aware systems. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HOTOS-IX)*, Lihei, Hawaii, 2003.
- [3] P. G. Bridges, W.-K. Chen, M. A. Hiltunen, and R. D. Schlichting. Position statement: Supporting coordinated adaptation in networked systems. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001.
- [4] P. G. Bridges, M. A. Hiltunen, R. D. Schlichting, and G. T. Wong. A configurable and extensible transport protocol, 2004. Submitted for publication.
- [5] R. Brightwell, A. B. Maccabe, and R. Riesen. On the appropriateness of commodity operating systems for large-scale, balanced computing systems. In *2003 International Parallel and Distributed Processing Symposium (IPDPS)*, Nice, France, April 2003.

- [6] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *Proceedings of the 2002 International Conference on Dependable Systems and Network*, pages 595–604, June 2002.
- [7] J. Fassino, J. Stefani, J. Lawall, and G. Muller. THINK: A software framework for component-based operating system kernels. In *Proceedings of the USENIX Annual Technical Conference*, 2002.
- [8] R. L. Graham, S.-E. Choi, D. J. Daniel, N. N. Desai, R. G. Minnich, C. E. Rasmussen, L. D. Risinger, and M. W. Sukalski. A network-failure-tolerant message-passing system for terascale clusters. In *Proceedings of the 16th International Conference on Supercomputing (ICS-16)*, pages 77–83, New York, New York, USA, 2002.
- [9] J. Jancic, C. Poellabauer, K. Schwan, M. Wolf, and N. Bright. dproc - extensible run-time resource monitoring for cluster applications. In *International Conference on Computational Science (2)*, pages 894–903, 2002.
- [10] T. Jones, W. Tuel, L. Brenner, J. Fier, P. Caffrey, S. Dawson, R. Neely, R. Blackmore, B. Maskell, P. Tomlinson, and M. Roberts. Improving the scalability of parallel jobs by adding parallel awareness to the operating system. In *Proceedings of SC'03*, 2003.
- [11] E. A. León, A. B. Maccabe, and R. Brightwell. Instrumenting LogP parameters in GM: Implementation and validation. In *Workshop on High-Speed Local Networks (HSLN)*, Tampa, FL, 2002.
- [12] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, May 2004.
- [13] J. Mellor-Crummey, R. Fowler, and D. Whalley. Tools for application-oriented performance tuning. In *Proceedings of the 15th ACM International Conference on Supercomputing (ICS-15)*, Sorrento, Italy, June 2001.
- [14] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall. The Paradyn parallel performance measurement tool. *IEEE Computer*, 28(11):37–46, 1995.
- [15] W. E. Nagel, A. Arnold, M. Weber, H. C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 12(1):69–80, 1996.
- [16] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pages 276–287, Saint Malo, France, 1997.
- [17] F. Petrini, D. Kerbyson, and S. Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q. In *Proceedings of SC'03*, 2003.
- [18] M. J. Sottile and R. G. Minnich. Supermon: A high-speed cluster monitoring system. In *IEEE Conference on Cluster Computing*, September 2002.
- [19] B. Tierney, W. E. Johnston, B. Crowley, G. Hoo, C. Brooks, and D. Gunter. The NetLogger methodology for high performance distributed systems performance analysis. In *HPDC*, pages 260–267, 1998.
- [20] J. S. Vetter. Dynamic statistical profiling of communication activity in distributed applications. In *Proceedings of SIGMETRICS 2002*, 2002.
- [21] K. Yaghmour and M. R. Dagenais. Measuring and characterizing system behavior using kernel-level event logging. In *Proceedings of the 2000 USENIX Annual Technical Conference*, 2000.
- [22] M. Zangrilli and B. B. Lowekamp. Comparing passive network monitoring of grid application traffic with active probes. In *Proceedings of the 4th International Workshop on Grid Computing (GRID2003)*, 2003.