# Effective HPC hardware management and Failure prediction strategy using IPMI

*Richard Libby*
rml@hpc.intel.com

## Abstract

Intelligent Power Management Interface (IPMI) defines common interfaces to "intelligent" hardware used to monitor a server's physical health characteristics, such as temperature, voltage, fans, power supplies and chassis. These capabilities provide information that enables system management, recovery, and asset tracking which help drive down the total cost of ownership (TCO) and increase reliability in today's HPC market. The new interfaces in IPMI v1.5 facilitate the management of rack-mounted HPC servers and systems in remote environment over serial, modem and LAN connections. New capabilities combined with the remote management functionality allow HPC IT managers to manage their servers and systems, regardless of system health, power state or supported communication media. IPMI compliant servers essentially eliminate the need for external hardware to perform the same function, thus saving costs. This paper will introduce the specification, the benefits of IPMI with respect to HPC and other clusters and how it could be used to generate alarms to a monitoring system before hardware failures become severe enough to cause cluster failure.

## 1 Introduction

If we were to look at what hinders large scale cluster deployments, only a handful of barriers come to mind. There is, of course, always the barrier of limited bandwidth, be it in the front side bus, the interconnect or any other form of I/O. Others include space, power, cooling, monitoring and management. This paper will focus on monitoring, management and predictive analysis: essentially overall HPC health and 'sickness prevention.' Intelligent Power Management Interface (hereafter referred to as IPMI) is an abstracted hardware layer that provides power control, alerting, sensor monitoring, Field Replaceable Unit (FRU) storage, Sensor Data Record (SDR) storage, customization, configuration all through multiple methods of secure access. The total cost of a hardware and software package has come down from US$30/node to sub-US$10, making management a viable solution to reduce the total cost of ownership (TCO).

The IPMI specification itself is a mere 450 pages, and the other supporting documents (four of them [Reference section: C, D, F & H], plus 23 referenced documents or RFCs) make for a good, long weekend of reading. This paper is a brief summary on how to use IPMI to get the most out managing large scale HPC deployments, and be able to use it to predict failures before they happen, thereby reducing the TCO of the entire system. The assumption is made in this paper that IPMI is implemented with strict consideration to the specifications.

## 2 Power Control

Hardware management of large scale HPC deployments today is quite possible if you have access to many people, infinite resources, or a magic wand. Since most of us have either none of these (in the case of the magic wand), or limited amounts of them (in the case of many people and infinite amounts of money), managing large scale deployments of anything is quite a chore. What do we really mean by 'managing?' Most think of managing a cluster as isolated systems: hardware and software. I would submit that it is a combination of both. If hardware fails, so does software. If software fails, hardware becomes a really expensive room heater. So, management becomes the ability to control software by looking at hardware as one failure point, or have the software control the hardware if thresholds increase beyond set limits. These control methods include powering off a server, powering on a server, resetting a server, and obtaining status of the power condition.

Resetting the server from a remote location seems trivial to most. "Why not just execute a command such a remote 'init 6' (Soft reset)?" or something similar depending on the operating system some might ask. Ah, but what happens if the operating system is hung? Oh, just send one of your people down and hit the reset button. In the case of the TeraGrid project, that may not be possible since there are at least four sites where the HPC might be at. So, a system reset function has gone from being a trivial action to a not-so-trivial action. Using IPMI, a system reset is trivial with a mere command.

Powering on and off a system is not quite as trivial to achieve. Sure it is, you say. You go out and purchase a network power switch, and spend about US$500.00 in the process. Ok, so you *can* achieve power control this way, but not cost effectively in a large scale HPC deployment. Let's say for example, that at best you could control eight machines with one network power switch. At US$63 per outlet, 256 outlets would cost you US$16k, 512 outlets would cost US$32k and 1000 outlets would cost US$63K. This adds only a little to the overall cost, but hey, if it is offered with an IPMI compliant server, you could use this additional money to purchase more servers. There's more on powering on and off, but it will be covered in another section.

In addition to being able to reset, power on and off, having the ability to query the server power status is also valuable. Instead of having to force power off (if on and vice versa), potentially losing a job or data, you can query to see if the server's power is on or off. This seems rather trivial at times, but in some instances when there is no OS available and the server is not in any proximity to you, it is nice to have this feature.

Lastly, one nice control feature is telling the UID light what to do, and how long to do it for. This is the little blue light found on the front and rear of most servers that turns on to identify it from other servers in cases of large scale HPC build-outs.

## 3 Alerting

Power control is essential, but from a predictive failure analysis standpoint, alerting is the driving force of determining if any one point of the HPC is suffering. Think of the alerting component as pain identification. This identification breaks down into event filtering, trapping (via SNMP), and policy management. Some aspects will not be mentioned fully, as they will be covered in other sections.

Without going into too much depth, when the Baseboard Management Controller (BMC, a.k.a. the service processor) observes an event

that occurs (see sensor section for types of events that might be interesting), it logs it into non-volatile space into a Server Event Log (SEL). Events generated either externally or internally are run through a filter within the BMC that can perform actions based on policies (more on policies later). These types of events might indicate impending danger of catastrophic events on any one machine, or in more serious cases, could indicate rack-wide implications, such as a fire. The ability to filter events combined with policy management gives the HPC administrator a powerful tool in predicting impending failure.

One of the keys to being able to predict (and later analyze) whether there is an impending failure, is knowing about it. The trapping component forwards on messages via SNMP to a "health master" that could look at the event (and compare it against a set of policies), determine if it was critical enough to ruin jobs on that node, and take appropriate action(s).

Policies can be set up to read either discrete or threshold-based values and define an action based on the type of event it is. For example, if a server in your HPC had a fan failure, the most likely thing you would see from an end user point of view is to hear the other fans ramp up. That action is an event that was driven by the BMC through a policy that was set either by a factory setting, or specified by an administrator programmatically. Once the other fans were ramped up to compensate for the missing fan, alerts can be trapped and sent via any number of methods to the "health master," which in turn, can determine (possibly by its own set of policies) if the event warrants moving data or jobs from one machine to a spare. Data can be recorded and analyzed later for recurring trends or even external environment conditions that might be causing failures. One example of this might be as follows: One rack might be sitting directly under a cooling source. The racks adjoining it would read higher ambient chassis temperature, and show skewed temperature data, but with the information, a stealthy sysadmin could correlate the HVAC 'on' times with a drop in ambient temperature inside the rack that gets hit with a blast of cool air. Although this condition in and of itself is not bad, one set of servers in the HPC might run faster due to being slightly cooler and cause other bottlenecks.

## 4 Common Sensor Model

So, what can IPMI in its present state touch and feel? Presently, it includes voltage, temperature, fan speeds, and presence.

Servers based on IPMI give the ability to monitor voltages on the baseboard and power supply. In cases where there is a lot of solar spot activity, this data could be useful. For example, too many voltage changes, and memory might start to have single bit errors, and too many of those might generate a multi-bit error that is not recoverable. From an industrial and commercial perspective, if the server suffered this, jobs would either be lost, or potentially have silent data corruption. Having the ability to monitor and log this data could be useful to see if there were external environment conditions that would impact server stability. In this particular example, a sysadmin might want to spec out Telco grade hardware that runs on DC power, thereby eliminating the external condition and increasing reliability.

With the tight thermal tolerances these days, a fan failure in server chassis can result in a catastrophic loss, especially in the smaller form factors (i.e. 1U). Having the ability to monitor fan speeds can greatly increase the chance of keeping jobs that are running on one member of the HPC and moving them to another, provided the event deems it worthy of

doing so. This does not just include chassis fans, but can be expanded to include processor, memory, and hard drive bay fans.

Temperature is a key aspect to monitor for predicting failure in an HPC. Logging events, using stats to show trends, and reacting to generated alerts are the ways to decrease a sysadmin's possibilities of downtime. It will also increase the ability to determine external environment conditions that would affect HPC performance, since heat is a critical aspect.

The last aspect of the common sensor model is the ability to detect presence. At times it would be nice to detect if the server chassis lid was opened. The system might log an entry into the SEL based on chassis intrusion. This might be useful information later on to determine causes of ambient temperature changes, or even to determine what time the processors and memory disappeared out of the best server.

## 5 Access Methods

Ok, so we are able to monitor all these cool things on our server, but how does the sysadmin or developer *really* get to them? There are multiple methods to get the information you need from an IPMI-based server. They include KCS, LAN, serial, modem, I2C, IPMB and ICMB. None of these methods will be discussed in too much detail here due to space and scope constraints.

One of the most common methods to get to the BMC is through an interface called Keyboard Controller Style (KCS). For IPMI laymen, this means a driver that is loaded and controlled through the OS. This is the preferred method to access the Intelligent Power Management Bus (IPMB), but certainly not the only one, and certainly in cases where the OS has gone south for the winter.

If the OS does go south, another nice method is using the Remote Management Control Protocol (RMCP) over the LAN. IPMI-based servers have 3.3 volts standby power that provides the BMC with enough juice to live on. With the cost of network interface cards (NIC) coming down, and many board manufacturers placing them on their server boards, this becomes the avenue on which to send UDP packets directed to port 623 that can issue IPMI commands. This becomes a close second to the KCS in that it replaces legacy serial concentration servers, allowing the sysadmin or IT manager to spend their US$50K (for every 40 ports and US$1.2M for 1000 servers) elsewhere on the HPC.

Other methods of access include direct serial using a serial concentration server. The downside to this interface is the cost for the serial concentration server. One nice feature worth mentioning in a Linux environment is Serial Over LAN (SOL). This allows the HPC sysadmin to enable serial redirect and have it piped over the network interface, thereby eliminating the need for a Keyboard-Video-Mouse controller (KVM). Costs are exorbitant and cabling becomes a nightmare for large scale HPC deployments. (Other features include private management bus (private I2C bus), and ability to add above board remote management cards.)

One nice feature of IPMI is the ability to forward information on to other IPMI compliant servers. Essentially, each IPMI server can become a proxy to relay messages to another IPMB through the Intelligent Chassis Management Bus (ICMB). A sysadmin or network architect could design an ICMB network (really an RS-845 multiport serial bus) that would allow up to 64 nodes per network. Management packets could be routed over this network instead of tying up the LAN channel, or in cases when the LAN was down, routing over ICMB. RS-845 also allows broadcasts, so the

sysadmin could invoke a broadcast to a bank of servers. Imagine it, a command that said: "Hello, rack of servers, shut down."

## 6 Authentication

All the methods of access are excellent to have, but how are they protected against intrusion? This section briefly discusses the channel privilege levels and encryption methods used to authenticate to the BMC.

There are four privilege levels to the BMC as defined by the IPMI specification: callback, user, operator, and administrative.

Callback is essentially irrelevant these days. It calls only a predefined number via a modem. It also only supports enough commands to initiate a callback. If the user is not at that location, then callback is not highly useful.

One nice channel privilege is called user. It would allow subsystems to monitor HPCs without being intrusive, and potentially lethal from a job perspective. Think of this privilege as 'sensor snag' only-essentially read-only or only benign commands are allowed. From a predictive analysis perspective, this level would give alerting mechanisms with the peace of mind that power control could be assigned to alternate members of the HPC.

Occasionally though, job scalability demands delegation of authority, but not total relinquishing of control. In these cases, the operator privilege level should be chosen. It has all commands available to administration, except configuration commands that can change the behavior of the out-of-band interfaces.

And then you have full rights-administrative privilege level. This privilege allows for full control over an IPMI-based server. In simple "health master" implementations, this would be the preferred privilege level to use. One quick word of caution: an administrative privilege level can even disable the channel they are coming in over.

Encryption is tied closely to authentication. There are basically three main points to remember on IPMI encryption. The first is that passwords can be sent clear-text, so from a security standpoint, it is something to consider. The second is that whatever the user initiates as an encryption algorithm is what will be returned, provided it falls into the third category. That is that there are generally two supported encryption algorithms, MD2 and MD5.

## 7 Field Replaceable Units (FRU)

An enterprise-class system will typically have FRU information for each major system board (e.g. processor board, memory board, I/O board, etc.). The FRU data can include information such as serial number, part number, model, and asset tag. What is this really good for in a failure prediction situation? Well, the "health master" could be programmed such that it could detect that a remote server's part is going to have a failure (let's just say overheating CPU), look up the part in a parts database, and alert the sysadmin which part they need to take with them to service the unit, what the unit serial number is, and possibly when it was last serviced. It would minimize downtime. By the way, FRU information can even be available when the system is powered down. Another useful example of FRUs being used is automated remote inventory. "IPMI does not seek to replace other FRU or inventory data mechanisms, such as those provided by SM BIOS, and PCI Vital Product Data. Rather, IPMI FRU information is typically used to complement that information or to provide information access out-of-band or under 'system down' conditions."

## 8 Sensor Data Records

IPMI was created with extensibility and scalability in mind. Unfortunately, with that capability comes a myriad of different components that can be monitored and controlled in different fashions. Sensor Data Records (SDRs) provide system management software the ability to retrieve information from the platform, and automatically configure itself to meet the capabilities of the platform-essentially an abstraction layer.

SDRs exist primarily to describe to server management software what a sensor configuration should look like, and to tell software to pay special attention to certain sub functions. SDRs are mostly not available to end users, but could potentially, especially if purchasing a board and chassis separately. For the most part, SDRs are made specifically for certain configurations of board/chassis combinations.

SDRs also define thresholds and actions based on those thresholds. For example, there might be an upper critical threshold on a board thermal sensor, and that might be tied via an SDR to an action of ramping up fans if excessive heat was detected. From a failure prediction standpoint, the "health master" could detect these changes, and ramp fans accordingly. This is just one small case, but you can see that the permutations get quite large with more servers in your HPC. What kind of sensor types are there? Two main categories exist-analog or digital, or fan, voltage, temperature. SDRs tell the BMC what the sensor type is in order to know how to process it (i.e. analog fan situation: you actually get a voltage back when polling it. The software will need to convert it via a mathematical function (a slope function: logarithmic, square root, quadratic, sin and many more defined in the IPMI spec.)

Realistically, the types of information that SDRs can store configuration on are: CPU sensors, chassis intrusion, power supply monitoring, fan speeds, fan presence, board voltages, board temperatures, bus errors, memory errors, and even possibly ASF progress codes (where the subsystem is in coming up-essentially a POST code). "Sensor Data Records are kept in a single, centralized non-volatile storage area that is managed by the BMC. This storage is called the Sensor Data Record Repository (SDR Repository). Implementing the SDR Repository via the BMC provides a mechanism that allows SDRs to be retrieved via 'out-of-band' interfaces, such as the ICMB, a Remote Management Card, or other device connected to the IPMB. Like most Intelligent Platform Management features, this allows SDR information to be obtained independent of the main processors, BIOS, system management software, and the OS."

## 9 System Event Log

Every IPMI compliant server has a System Event Log (SEL) which is a centralized, non-volatile repository for all events generated. Think of this not only as a BMC journal. Any authenticated user can enter a SEL entry.

Common events that might be stored are reboots, processors offline, memory (both single and multiple bit) errors, sensors that go beyond set thresholds, PCI parity errors (PERR), Non-maskable interrupts (NMI), and many more. The "health master" could periodically poll select system event logs from the HPC, and determine if there are potential problems that could be coming down the wire, and mitigate a response to those problems.

Alerting is done off the SEL as well. When an event is written to the SEL, it is checked against policies and threshold values, and performs actions based on those policies. Too

much of a health conscious sysadmin could put in place tighter thresholds, and decrease the possibility of downtime, while at the same time possibly risking more time checking out false alarms. A truly health conscious sysadmin would determine standard thresholds to work with safely, while keeping in mind that a false alarm here and there might keep them on their toes and possibly prevent a catastrophic failure on a node (or more in some cases).

Clearing the SEL is possible as well, in fact, once a SEL is queried and data is stored on the "health master," it is advisable to clear the SEL. The main reason for this is that if the SEL fills up, new entries are dropped. This could cause a sysadmin to miss critical events that could lead either to extensive downtime, or a catastrophe.

## 10 Configuration

IPMI gives such flexibility that many aspects can be configured to fit the end-user's needs.

These parameters include: BMC policies, LAN configuration (such as static or dynamic IP address, mask, and gateway), privilege levels, alerting functions (such as whom to forward alerts to), sensor polling rates, etc.

Imagine for a moment that your "health master" detects that it one of the nodes in the HPC is going down due to some failure, and you are able to retrieve the critical data off the hard drive. When you bring up a spare node in its place, you can dump the hard drive data back down, right? Sure, but what about BMC configuration? This is where IPMI shines through if implemented properly. With most IPMI compliant servers today, a sysadmin can retrieve the BMC information. But stuffing it back onto another machine is a little trickier, especially if it is over the LAN interface.

## 11 Questions Users Might Ask

A few questions might remain in the reader's mind still after this brief whetting. One such question might be, "What is the relationship or difference between IPMI and Alert Standard Forum (ASF)?" "While somewhat of an oversimplification, ASF may be considered to be scoped for 'desktop/mobile' class systems, and IPMI for 'servers' where the additional IPMI capabilities such as event logging, multiple users, remote authentication, multiple transports, management extension busses, sensor access, etc., are valued. However there are no restrictions in either specification as to the class of system that the specification can be used. [(i.e.)] you can use IPMI for desktop and mobile systems and ASF for servers if the level of manageability fits your requirements."

Another might be, I don't have IPMI capable servers today, can I add in an IPMI card? The short answer to that question is possibly, but it depends on your server base board. Provided it has an interface to IPMB, the possibilities increase. But, the best thing to do is to weigh the costs associated with not having IPMI (as mentioned earlier-no KVM, network power switch, serial concentration server) and see what benefits it can bring to predict an impending failure before it happens.

## 12 Summary

The key to using IPMI for failure prediction analysis is the polling and listening software, and how intelligent it is at analyzing the data to make predictions as to where problems lie.

As most HPC sysadmins know, one large scale cluster deployment barrier is management, monitoring, fault isolation and failure prediction. IPMI enables a sysadmin a great way to reduce the TCO of HPCs by monitor-

ing overall HPC health and 'prevent sickness' by providing an abstracted hardware layer that provides power control, alerting, sensor monitoring, Field Replaceable Unit (FRU) storage, Sensory Data Record (SDR) storage, customization, configuration all through multiple methods of secure access-all through software.

## 13 Call to Action

When writing RFQs, make sure to include IPMI as a required feature in order to reduce TCO and increase manageability in HPC deployments.

## 14 References


A. Alert Standard Format v1.0 Specification, ©2001, Distributed Management Task Force. `http://www.dmtf.org`

B. The I2C Bus And How To Use It, ©1995, Philips Semiconductors. This document provides the timing and electrical specifications for I2C busses.

C. Intelligent Chassis Management Bus Bridge Specification v1.0, rev. 1.2, ©2000 Intel Corporation. Provides the electrical, transport protocol, and specific command specifications for the ICMB and information on the creation of management controllers that connect to the ICMB. `http://developer.intel.com /design/servers/ipmi`

D. Intelligent Platform Management Bus Communications Protocol Specification v1.0, ©1998 Intel Corporation, Hewlett-Packard Company, NEC Corporation, and Dell Computer Corporation. This document provides the electrical, transport protocol, and specific command specifications for the IPMB. `http://developer.intel.com /design/servers/ipmi`

E. Intelligent Power Management Interface Specification v1.5; rev. 1.1, ©2002 Intel Corporation, Hewlett-Packard Company, NEC Corporation, and Dell Computer Corporation. `http://developer.intel.com /design/servers/ipmi`

F. IPMI Platform Event Trap Format Specification v1.0, ©1998, Intel Corporation, Hewlett-Packard Company, NEC Corporation, and Dell Computer Corporation. This document specifies a common format for SNMP Traps for platform events.

G. Proposal for Callback Control Protocol (CBCP), draft-ietf-pppext-callback-cp-02.txt, N. Gidwani, Microsoft, July 19, 1994. As of this writing, the specification is available via the Microsoft Corporation web site: `http://www.microsoft.com`

H. Platform Management FRU Information Storage Definition v1.0, ©1999 Intel Corporation, Hewlett-Packard Company, NEC Corporation, and Dell Computer Corporation. Provides the field definitions and format of Field Replaceable Unit (FRU) information. `http://developer.intel.com /design/servers/ipmi`

I. RFC 1319, The MD2 Message-Digest Algorithm, B. Kaliski, RSA Laboratories, April 1992.

J. RFC 1321, The MD5 Message-Digest Algorithm, R. Rivest, MIT Laboratory for Computer Science and RSA Data Security, Inc. April, 1992.

K. System Management BIOS Specification, Version 2.3.1, ©1997, 1999 American Megatrends Inc., Award Software International, Compaq Computer Corporation, Dell Computer Corporation, Hewlett-Packard Company, Intel Corporation, International Business Machines Corporation, Phoenix Technologies Limited, and SystemSoft Corporation.

L. System Management Bus (SMBus) Specification, Version 2.0, ©2000, Duracell Inc., Fujitsu Personal Systems Inc., Intel Corporation, Linear Technology Corporation, Maxim Integrated Products, Mitsubishi Electric Corporation, Moltech Power Systems, PowerSmart Inc., Toshiba Battery Co., Ltd., Unitrode Corporation, USAR Systems.

M. The TeraGrid Project, National Center for Supercomputing Applications, San Diego Supercomputer Center, Argonne National Laboratory and Pittsburgh Supercomputing Center; http://www.teragrid.org

N. Wired for Management Baseline Version 2.0 Release, ©1998, Intel Corporation. Attachment A, UUIDs and GUIDs, provides information specifying the formatting of the IPMI Device GUID and FRU GUID and the System Management BIOS (SM BIOS) UUID unique IDs.

## 15 Glossary of Terms

**BMC** Baseboard Management Controller

**CMOS** In terms of this specification, this describes the PC-AT compatible region of battery-backed 128 bytes of memory, which normally resides on the baseboard.

**Diagnostic Interrupt** A non-maskable interrupt or signal for generating diagnostic traces and ‘core dumps’ from the operating system. Typically NMI on IA-32 systems, and an INIT on Itanium®-based systems.

**EFI** Extensible Firmware Interface. A new model for the interface between operating systems and platform firmware. The interface consists of data tables that contain platform-related information, plus boot and runtime service calls that are available to the operating system and its loader. Together, these provide a standard environment for booting an operating system and running pre-boot applications.

**FRB** Fault Resilient Booting. A term used to describe system features and algorithms that improve the likelihood of the detection of, and recovery from, processor failures in a multiprocessor system.

**FRU** Field Replaceable Unit. A module or component which will typically be replaced in its entirety as part of a field service repair operation.

**Hard Reset** A reset event in the system that initializes all components and invalidates caches.

**HPC** High Performance Computing, commonly computationally intense.

**I2C** Inter-Integrated Circuit bus. A multi-master, 2-wire, serial bus used as the basis for the Intelligent Platform Management Bus.

**ICMB** Intelligent Chassis Management Bus. A serial, differential bus designed for IPMI messaging between host and peripheral chassis. Refer to [ICMB] for more information.

**I/O** Input / Output. Typically refers to I/O subsystems such as PCI (and variants), memory, and CPU buses.

**IPM** Intelligent Platform Management.

**IPMB** Intelligent Platform Management Bus. Name for the architecture, protocol, and implementation of a special bus that interconnects the baseboard and chassis electronics and provides a communications media for system platform management information. The bus is built on I2C and provides a communications path between ’management controllers’ such as the BMC, FPC, HSC, PBC, and PSC.

**NMI** Non-maskable Interrupt. The highest priority interrupt in the system, after SMI. This interrupt has traditionally been used to notify the operating system fatal system hardware error conditions, such as parity errors and unrecoverable bus errors. It is also used as a Diagnostic Interrupt for generating diagnostic traces and ‘core dumps’ from the operating system.

**MD2** RSA Data Security, Inc. MD2 Message-Digest Algorithm. An algorithm for forming a 128-bit digital signature for a set of input data.

**MD5** RSA Data Security, Inc. MD5 Message-Digest Algorithm. An algorithm for forming a 128-bit digital signature for a set of input data. Improved over earlier algorithms such as MD2.

**PEF** Platform Event Filtering. The name of the collection of IPMI interfaces in the IPMI v1.5 specification that define how an IPMI Event can be compared against 'filter table' entries that, when matched, trigger a selectable action such as a system reset, power off, alert, etc.

**PERR** Parity Error. A signal on the PCI bus that indicates a parity error on the bus.

**PET** Platform Event Trap. A specific format of SNMP Trap used for system management alerting. Used for IPMI Alerting as well as alerts using the ASF specification. The trap format is defined in the PET specification. See [PET] and [ASF] for more information.

**POST** Power On Self Test.

**RFQ** Request for Quote.

**SDR** Sensor Data Record. A data record that provides platform management sensor type, locations, event generation, and access information.

**SEL** System Event Log. A non-volatile storage area and associated interfaces for storing system platform event information for later retrieval.

**SERR** System Error. A signal on the PCI bus that indicates a 'fatal' error on the bus.

**Soft Reset** A reset event in the system which forces CPUs to execute from the boot address, but does not change the state of any caches or peripheral devices.

**TCO** Total Cost of Ownership. All of the possible costs involved in the purchase, installation, management, support and use of the IT infrastructure within an organization throughout a product's life cycle, from acquisition to disposal.

# Proceedings of the Linux Symposium

July 23th–26th, 2003
Ottawa, Ontario
Canada

## Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

## Review Committee

Alan Cox, *Red Hat, Inc.*
Andi Kleen, *SuSE, GmbH*
Matthew Wilcox, *Hewlett-Packard*
Gerrit Huizenga, *IBM*
Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*
Martin K. Petersen, *Wild Open Source, Inc.*

## Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*