PRISM

A supercomputing real-time VaR engine
based on a Linux cluster

A Mercé Technical Report

TR-2001-15

Merce Technologies Private Limited
merceworld.com
Mumbai

Contents

1 The objective

The National Stock Exchange of India Limited (NSE), located in Mumbai, is one of the top five stock exchanges in the world in terms of number of trades per hour. This stock exchange, which is also one of the most modern in terms of information systems, uses a computerised open order book on high-end fault-tolerant hardware and a satellite-based nationwide WAN with 3000+ trading terminals connected online. The NSE never had an open-outcry trading ring.

India has the second largest number of stock owners and listed companies worldwide, after the US. The NSE is India's largest stock exchange, with 2,500+ companies listed. Daily trading volumes on the NSE in cash, futures and options are several billion US dollars.

In 1999, the NSE decided to explore the possibility of a real-time risk analysis system to track the risk exposure of individual brokers and investors while trades were being generated in the exchange system. The NSE wanted to use the internationally accepted VaR (Value at Risk) algorithm with at least 5,000 Monte Carlo iterations for each trade, and recalculate exposure for each of the two parties in each trade in real time. This would need supercomputer level floating point performance, coupled with fault tolerance, given the fact that the NSE infrastructure today handles more than one hundred trades per second.

Our solution for the NSE is called PRISM — the Parallel Risk Machine. It performs real time risk analysis of the entire NSE trading activity.

2 The project team

PRISM was built by a joint team comprising Merce Technologies (then called Starcom Software Private Limited), The Indira Gandhi Institute of Development Research (IGIDR, www.igidr.ac.in), Infotech Financials Pvt Ltd (www.infofin.com) and other consultants.

Mercé was the technical lead for technology and tools choices, including the critical choice of cluster computing software. We also designed and built the framework software, the messaging layer and fault-tolerant cluster computing platform, using industry-standard MPI (Message Passing Interface) on Linux and Digital Unix systems. Getting the desired super-computer throughput and demonstrating fault tolerance in the compute back-end was Mercé's responsibility.

Professors Ajay Shah (ajayshah@igidr.ac.in) and Susan Thomas (susant@igidr.ac.in) of IGIDR were the architects for the econometrics components of PRISM and

headed a team of researchers at IGIDR which provided the high-performance VaR code for it.

A team from Infotech Financials worked on the UID (the User Interface Daemon), which was written as a platform-independent GUI application in Java, and handled display of information from the supercomputer core in near real-time, without loading the compute cluster. It also allowed an administrator to do drill-down queries into the risk positions of various market entities, and change risk limits of any of them. Some of the other key components were also developed by the Infofin team.

Avadhoot Deshpande worked as a consultant and coordinated the various technical teams, interacted with the NSE, worked out details of the CTCL feed (real time two-way data link with NSE's trading system), and tuned the user-end features to fit in with NSE's requirements. His prior experience with NSE's CTCL link was very useful in successful integration of PRISM with the NSE trading system.

Some additional information about PRISM can be found on Infofin's Website, at http://www.infofin.com/INFOFIN/prism.htm.

3 The architecture of PRISM

PRISM is built on a fault-tolerant cluster of inexpensive Intel Linux systems connected by an off-the-shelf 100 MHz switched Ethernet. This system provides the required performance at a tiny fraction of the price and TCO of a supercomputer. The system is live and in production since 2000 at Mumbai. The architecture has been tested to be scalable beyond 1,000 trades per second, and can scale further purely depending on computing hardware and network specifications.

There are many communicating computers that comprise the PRISM cluster (see Figure 1). The hub of PRISM is the computer labelled as "mother node" in the figure. In addition, there are a number of computers running "child" processes or "monks" for handling the actual numerical computation load, and various other computers with large graphical display screens, running the UI application for management and monitoring of PRISM. The current deployment of PRISM at NSE connects the cluster using a 100 Mbits/sec Ethernet switch. Future deployments to handle more than 1,000 trades/sec may require higher-end interconnect hardware; the software should not need any change at all. Unless otherwise mentioned, all the code in PRISM is written in C, and will work on any modern flavour of Unix, Linux, FreeBSD, *etc.*, provided it supports a recent, compatible version of MPI and other system libraries.

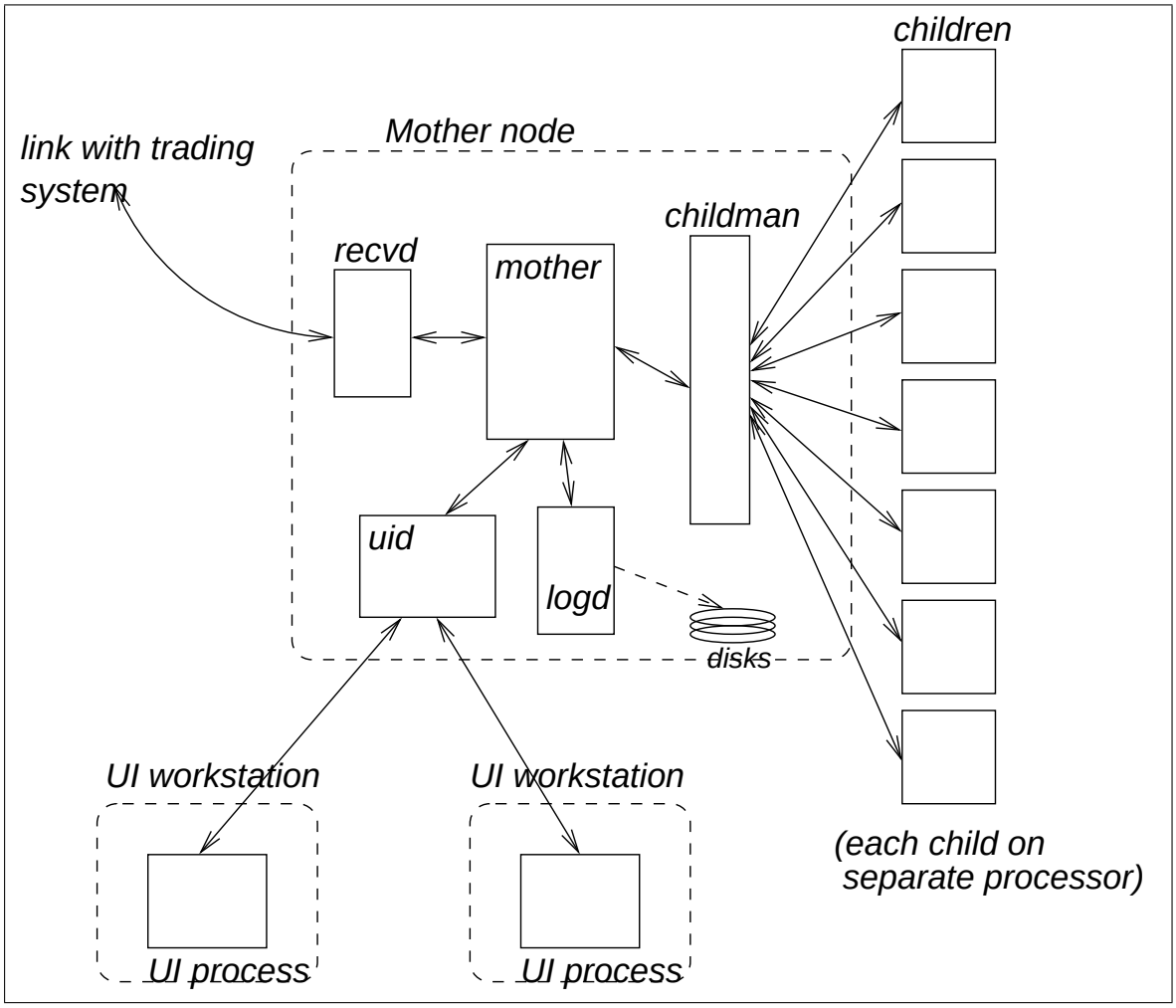

Figure 1: The PRISM architecture with communicating processes

PRISM comprises multiple processes communicating among themselves, executing on these computers. Almost all communication among these processes is done using the high performance, low latency messaging protocol provided by MPI. *mother* is the most important process in PRISM, and executes on the "mother node," which today is a multiprocessor Digital Alpha server running Digital's Tru64 Unix. Each child node has one or more *child* processes. NSE has made available dual-CPU Intel Pentium III computers for the child nodes of this installation. Therefore, we have configured each child node to run two *child* processes, one for each CPU. Thus, for this installation of PRISM, there are exactly twice as many compute processes as physical computers for child nodes. Each *child* process executes a copy of the VaR algorithm developed by the team from IGIDR.

The interaction with the external world is through an intuitive and sophisti-

cated GUI, which translates all the dynamic and structural complexity of the system into paradigms that a capital market regulator or economist can easily relate to. The UI is implemented as a Java application. This makes it easy to port the UI application to any OS or hardware. This Java application is shown as "UI process" in the figure. No MPI has been used in the UI process; the communication between the UI processes and *uid* uses a separate, simple custom-designed protocol based on TCP/IP. There can be any number of instances of the UI processes, thus permitting any number of "consoles" onto PRISM, distributed across a TCP/IP network of any size.

4 Principles of operation of PRISM

4.1 Startup and shutdown

PRISM starts up before the beginning of trading hours on every working day, and is shut down after end of trading hours.

PRISM needs to maintain a local copy of the stock portfolio of each investor, Trading Member (*i.e.* broker), and Clearing Member (*i.e.* member of the Clearing Corporation). This information is maintained in in-memory data structures in *mother*. At startup, *mother* reads a file stored on the file system of the "mother node" and initialises its data structures. At shutdown, *mother* writes back the current snapshot to disk.

At startup, the bootup process also fires *child* processes, does health checks of other processes, and validates connectivity among nodes.

4.2 Normal operation

After startup, the feed from the trading system begins pumping in trade information into PRISM. This information is in NSE's custom format with its own idiosyncracies, and needs to be validated, decoded for machine dependencies — the format is hardware byte-order sensitive, for instance — and translated into a format that can then be processed cleanly by the rest of PRISM. This is done by *recvd*, the "receive daemon." *recvd* also splits each trade message into two half-trade messages, *i.e.* it splits the information about a trade into one message about the buyer and another about the seller. PRISM is only concerned with half-trades, because its analysis of risk exposure of a party does not depend upon the counterparty of the trade.

From *recvd*, half-trade information flows to *mother*. This is the first hop of the flow that uses MPI.

mother receives each half-trade message and updates portfolio positions of

the corresponding buyer or seller in its in-memory data structures. This also changes the portfolio and settlement liability of the corresponding Trading Member and Clearing Member. Each of these new positions is then packaged, together with some market-wide indicators and computed results, and sent for VaR analysis in a message from *mother* to *childman*.

childman is "child manager;" its job is to manage the children, which are pure compute servers. *childman* is a scoreboarding process, which keeps track of which *child* processes are free. When *childman* receives a VaR request message from *mother*, it picks up the next free *child* from its free list, and sends the VaR request to it. It then internally marks that *child* as busy, and records the timestamp of the despatch of the request. When the results of the computation come back from the *child*, *childman* marks the *child* as free once again, and returns the results to *mother*. If *mother* detects that any entity has exceeded risk limits, it sends a message back *via* *recvd* to the trading system, requesting disabling of the entity.

childman has another critical function; it holds the intelligence which implements partial fault tolerance in PRISM. This is described in Section 5 below.

child processes have no residual state; all inputs required for a VaR computation are sent to it by *mother* *via* *childman* in its request message. They do not need to record any information on disk, or remember any information other than messaging housekeeping information in memory. In fact, *child* processes do not access the hard disks on their computers after bootup, and can be effectively deployed on diskless computers provided there is adequate RAM. Their usage of RAM follows a predictable pattern and does not grow indefinitely; this makes it possible to deploy hardware with adequate RAM and eliminate the possibility of even page faults and swapping.

mother also communicates with *uid* and *logd*. *logd* — the "logging daemon" — has a simple function: it logs each and every significant action performed in PRISM to a perpetually growing file on disk. It receives a stream of non-blocking MPI messages from *mother*, thus avoiding slowing down the core computation processes in the event of a random slowdown or error in the disk subsystem. *logd* is therefore a sink of information; it never passes back any messages to *mother*. The logging feature is an audit requirement of the market regulator.

uid, or the "UI daemon" manages the user interfaces processes, which, as has already been described, are written in Java. One of the functions of *uid*, therefore, is to act as a bridge between the custom-built network protocol used by the Java applications and the internal MPI messaging layer of the rest of PRISM. Another important function of *uid* is to insulate *mother* from any synchronous connection with the relatively unstable and unpredictable behaviour of the Java applications which are managed by humans; any sud-

den slowdown of one Java process in the middle of its communication will not affect a waiting *mother* in the current design, because *uid* will buffer it. A third function of *uid* is to act as a multiplexor, taking a single data stream from *mother* and feeding it to any number of UI processes. The largest part of the communication from the core of PRISM to its UI processes is the steady stream of one-way messages carrying information of each half-trade, each risk position calculation, and each event (*e.g.* disablement of an investor). These messages are sent out by *mother* to *uid* as one stream using low-latency MPI messages, and then replicated by *uid* to as many streams as there are UI processes using simpler, higher latency protocols.

5 Fault tolerance in PRISM

In the current version of PRISM, there is transparent fault tolerance implemented against any number of failures of *child* computers or processes. This means that if there are N *child* processes, then upto $N - 1$ of these can fail at any time, and PRISM will continue to function with only small delays — of sub-second order of magnitude — at the precise instant of each failure, followed by proportional degradation of throughput. PRISM is said to have partial fault tolerance because there is not yet any fault tolerance for failure of *mother*, *childman*, *etc.* The UI is vestigial; PRISM can function from beginning to end of trading hours without any UI process.

childman constantly monitors the round-trip times taken by the *child* processes to return with VaR results.[1] If it finds that a busy *child* has taken three times the average round-trip time and has not yet returned with a result, *childman* simply marks that *child* as unavailable and re-assigns the VaR computation to another free *child*. It then passes this information on to *mother* as a *child*-failure message, so that *mother* can report it *via* *uid* to the user interfaces, and log it *via* *logd*. *childman* never assigns any future communication to an unavailable *child*.

Thus, this simple approach allows PRISM to continue processing risk exposure information, albeit at reduced throughput, as long as at least one *child* process is active. A computer in the cluster running *child* processes can be switched off at any time without warning, without disrupting PRISM.

Future enhancements will probably include a messaging arrangement by which a system administrator can issue a command to PRISM *via* its UI to re-consider a "corrected" *child* and add it to its pool. A more challenging task will be moving from partial to complete fault tolerance.

[1] In the current system, this time is less than ten milliseconds.

6 Conclusion

PRISM is a very sophisticated, high-performance solution which incorporates cutting-edge econometrics as well as cutting-edge cluster computing technology for a real-life business solution whose usefulness can easily be justified by an analysis of its RoI. There is no comparable system to PRISM in existence at the time of this writing.

For more information and inquiries, contact:

Merce Technologies Private Limited
phone: +91 22 4153 0500
email: sales@merceworld.com

* * * * *