Supermon: High Speed, Scalable Cluster Monitoring

Matt Sottile (matt@lanl.gov)
Los Alamos National Laboratory,
Advanced Computing Laboratory

Scalable Systems Software Meeting, 21-22 February 2002

Outline

1. Introduction
2. Architecture
3. Performance
4. Applications and Future Work
5. Conclusion

Introduction

Monitoring: The act of observing a system via a set of sensors.

- *Hardware vs Software monitoring*: Tradeoffs in perturbation, system complexity, and cost.
- *Reactive vs Periodic monitoring*: Monitoring in special circumstances vs. continuous sampling at fixed intervals.

The emphasis in supermon is balancing high-sampling rates with minimal perturbation of applications.

Perturbation (*or, not monitoring the monitor*)

Given a variable X that we sample:

$$X_{obs} = X_{actual} + X_{error}$$

X_{error} is introduced by the monitoring software.
Let $\epsilon << X_{actual}$ be the maximum tolerable error. If I is a measure of the *intrusiveness* of the sensor, and X_{error} is a function of I, then we want:

$$|X_{error}(I)| < \epsilon$$

Since I is related to the sampling rate, the goal with supermon is to allow the maximal sampling rate s.t. the error is still less than ϵ. This drove most (all?) of the architecture design decisions.

Supermon architecture

Supermon is broken down into four distinct components:

1. A loadable kernel module providing data
2. The “mon” single node data server
3. The “supermon” data concentrator
4. Clients

Symbolic expressions form the basis of the protocol binding each of the four layers.

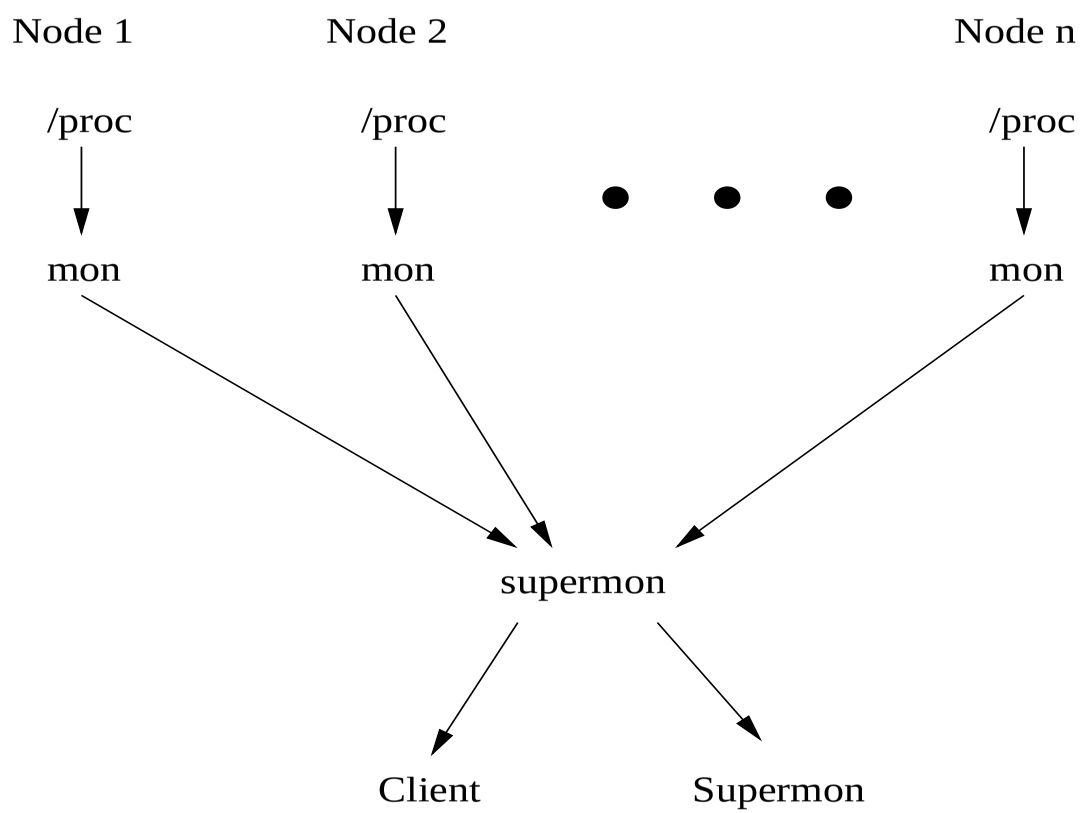

Figure 1: Architecture illustration.

Symbolic expressions and supermon

S-expressions were introduced in the 1950s with LISP.

```
sexpr ::= ( element )

element ::= atom etail | sexpr etail | ϵ

etail ::= element | sexpr | ϵ
```

Complex data is *easy* to encode in s-expressions, as is meta-data.

Example: A tree

```
(nodeA
  (nodeB
    (node C) ()
  )
  (nodeD)
)
```

Loadable Kernel Modules and `/proc`

The loadable kernel module under Linux provides two additional entries in `/proc` for supermon data to be retrieved by clients:

- `/proc/sys/supermon/#` : This contains a description of the machine.
- `/proc/sys/supermon/S` : This contains the data reflecting the state of the machine at the time it is read.

`/proc/sys/supermon/#`

(cpuinfo (nr 4) (user nice system))
(avenrun (nr 1) (avenrun0 avenrun1 avenrun2))
(paging (nr 1) (pgpgin pgpgout pswpin pswpout))
(switch (nr 1) (switch))
(time (nr 1) (timestamp jiffies))
(netinfo (nr 6) (name rxbytes rxpackets rxerrs rxdrop rxfifo rxframe rxcompressed rxmulticast tx bytes txpackets txerrs txdrop txfifo txcolls txcarrier txcompressed))

* *nr indicates the "arity" of the variables for the given category.*

`/proc/sys/supermon/S`

(cpuinfo (user 44042292 88026439 64765636 87093318) (nice 17193985 11936486 17778150 19162835) (system 0 0 0 0))
(avenrun (avenrun0 102) (avenrun1 373) (avenrun2 354))
(paging (pgpgin 174688564) (pgpgout 192768264) (pswpin 15) (pswpout 408))
(switch (switch 768964692))
(time (timestamp 0xebdf04f8b7) (jiffies 0x3b6b37ef))
(netinfo (name lo eth0 eth1 eth2 myri0 myri1) (rxbytes 1425399699 5815331027 0 53131479234 21234062 0) (rxpackets 5084174 8924076 0 235733640 505571 0) (rxerrs 0 0 0 0 0 0) (rxdrop 0 0 0 0 0 0) (rxfifo 0 0 0 0 0 0) ...)

mon : Providing single-node data via TCP

- Mon is a simple server that runs on a node and serves data from `/proc` to clients via TCP sockets.
- The data format between mon and clients is a slightly modified version of the `/proc` format containing structure necessary for scalability and composition.
- Mon maintains per-client filters (using bitmasks) to allow clients to request subsets of the available data.

(cpuinfo *(node 0x12344556) (mask 0x1) (nm (1 (0x12344556))) ...*)

supermon : The data concentrator

- Supermon connects to multiple mon clients and combines their data streams into a single stream for clients.
- Asynchronous socket code improves performance to avoid the bad effects of slow or dead clients.
- "Speaks" the same protocol to both clients and mon servers, allowing supermon servers to speak to other supermon servers.
- Currently a major focal point of research for improving maximum sampling rates: supermon topologies, data filtering and reduction, etc...

Clients

- Clients are simple: as long as they can understand s-expressions, they can use the data for any purpose.
- Clients determine the sampling rate - if a client does not send a request, then mon, supermon, and `/proc` are idle and sleep.
- Meta-data (# *command*) is available at any layer of supermon, so a client can use many data sources with no modification.

Later we give example client applications...

Performance of supermon

The performance of supermon looks at the number of samples we can achieve in one second - how many Hz can we sample at?

We look at the performance of supermon at multiple levels.

1. `/proc` to a client.
2. `/proc` to a mon to a client
3. `/proc` to a mon to a supermon to a client.

For information on the testbed used for measuring supermon performance, please refer to "Life with Ed", HPDC '02.

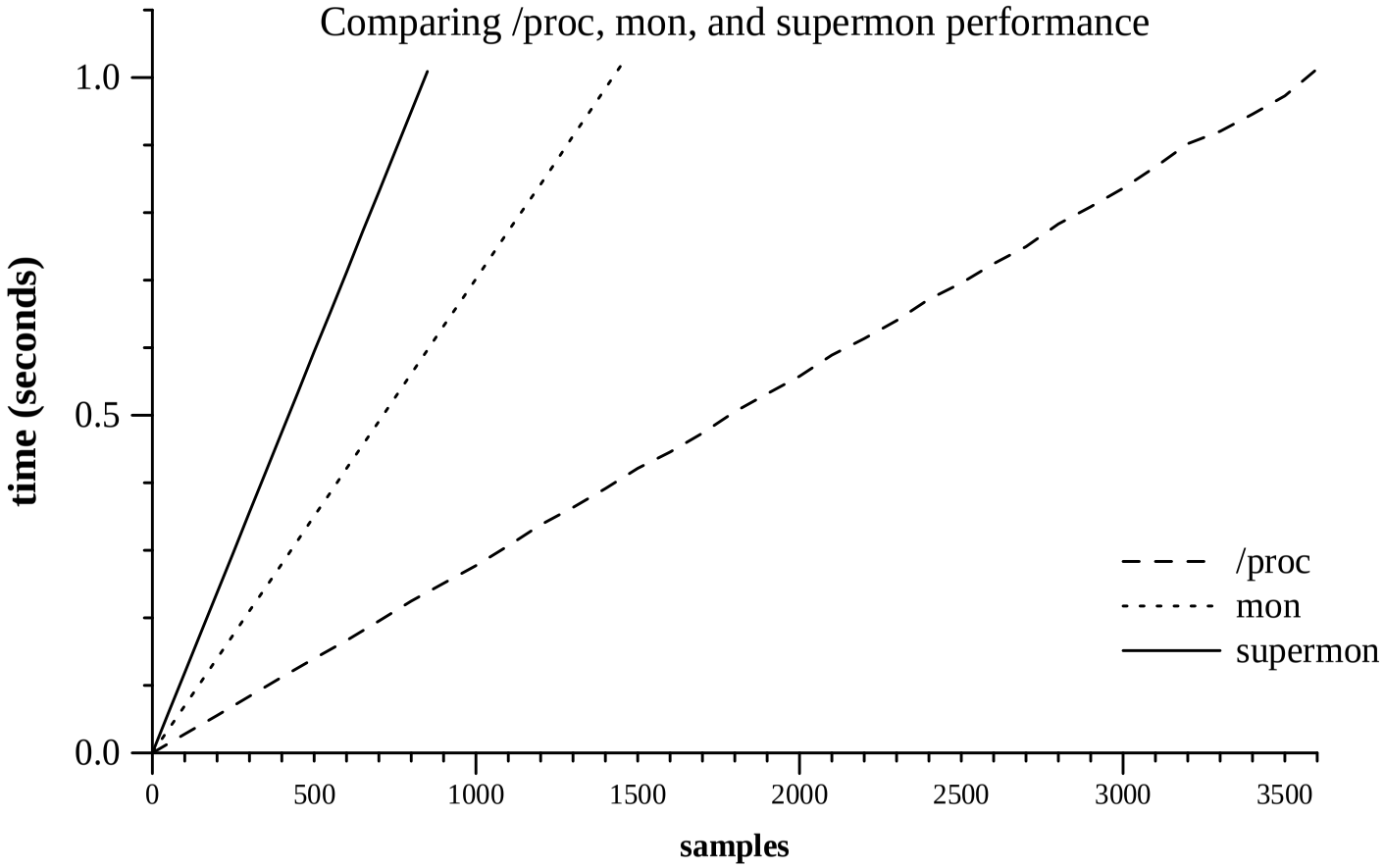

Figure 2: A comparison of the layers : supermon, mon, `/proc`

Flashback: The old way to sample single node data

The following plot is simply shown to prove that our new technique is *much faster* than the old method used to gather data from a single node.

* The old way to sample is equivalent to reading once directly from the `/proc` entry provided by the supermon kernel module.

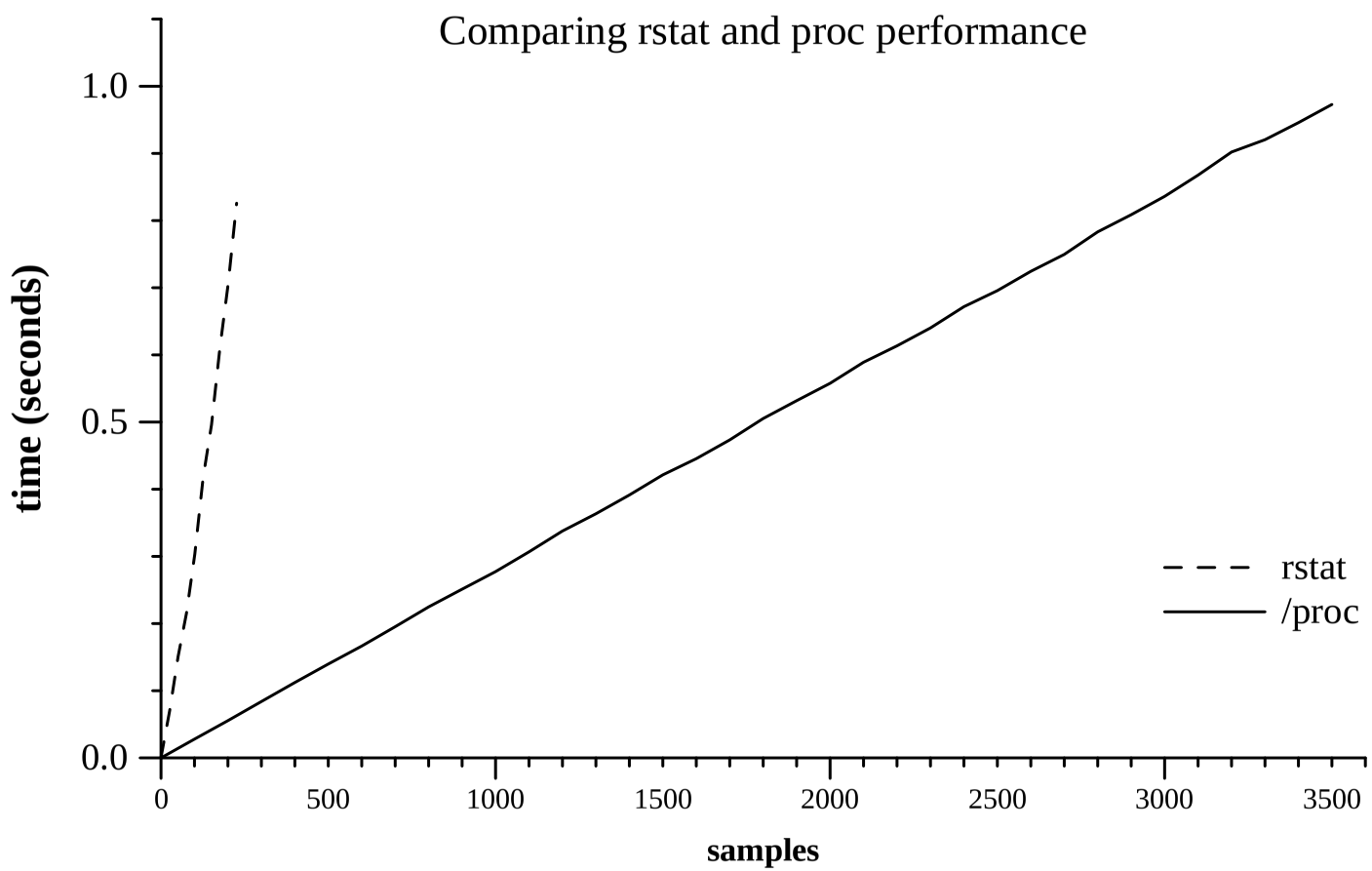

Figure 3: Sampling with the old (rstat) vs new (`/proc`) methods.

Supermon performance: Some numbers...

The peak sampling rates we observe at each layer:

- `/proc` : 3500Hz
- mon : 1400Hz
- supermon : 750Hz

We improved over older methods also:

- `/proc` vs RPC.RStatd : 3500Hz/275Hz yields 12x improvement

This is good - we can sample through three layers (`/proc` to supermon) faster than the original rstat could at the lowest layer! Note that each measurement involved samples containing *ALL possible data*, which is a larger data set than rstat provided.

Performance continued...

But who wants to monitor *just one node*?!

Nodes	*Sampling Rate*
5	400Hz
10	225Hz
20	125Hz
100 *Flat*	66Hz
100 *10-node fanout*	57Hz
100 *50-node fanout*	35Hz

Table 1: Scaling results for Supermon.

Applications of supermon

- Failure prediction and intelligent application reaction
- Algorithm visualization
- Performance analysis
- Rapid identification of failed components
- Improved system state presentation via `/proc`
- Scheduling tools (*bproc in progress*)

Future work

1. Debug, debug, debug...
2. Integrate hardware sensors (temperature, fan speeds, voltages)
3. Write clients for users who don't want to write their own
4. Integrate application level monitoring data (such as TAU)
5. Data analysis techniques - non-trivial problem, particularly in "realtime".

Conclusion

- Supermon is *fast*.
- Even through multiple levels of filtering and network transport, Supermon is faster than existing monitoring tools.
- S-expressions are significantly better than custom protocols if we want general tools.
- Supermon has already revealed features in cluster computing that are very interesting (MPI behavior, actual benefit(*?*) of heirarchy, limits of Linux).

For more information...

Contact info: supermon@lanl.gov
Web page: http://www.acl.lanl.gov/supermon/

For performance and architectural details, a copy of the paper "Supermon: Cluster Monitoring as if Performance Mattered", (submitted to ICS'02), can be provided upon request. Also, current version of code only available by request.