Introduction

Online or Real-time application monitoring allows tracking performance characteristics during execution as opposed to doing so post-mortem. This opens up several possibilities otherwise unavailable such as real-time visualization and application performance steering. These enable adaptation and control of long-running applications.

Two fundamental components that constitute an online application performance monitor are the measurement and transport systems. The former captures performance metrics of individual contexts (processes, threads). The latter enables querying the distributed/distributed state from the different contexts. The transport may also be used to control the measurement system. As HPC systems grow in size and complexity, the scalability and overhead of the online performance monitor become increasingly important considerations.

Approach and Rationale

We adapt two existing, mature systems - Tuning and Analysis Utility (TAU) and Supermon - to solve the problem of scalable, low-overhead online performance monitoring of parallel applications. TAU performs the role of the measurement system and is adapted to use Supermon as the transport from which sinks query the distributed performance state. Alternate approaches could have been to use the traditional method of performance data transport, namely using a network filesystem or to invent a new transport completely within TAU.

The rationale behind our current approach is as follows: i) Using a traditional transport incurs high overhead (as the graphs demonstrate). ii) Keeping the transport outside TAU (or even performance monitors) that want to use it. iii) No new components are added to the system as a cluster-monitor is typically present already. iv) This allows close correlation between system-level information (such as from the OS and hardware sensors) with application performance as the cluster-monitor already reports the former and v) Lastly, Supermon is lightweight and scalable due to its hierarchical structure.

Supermon Cluster Monitor

Supermon is a scalable monitoring system for high performance computing systems. Its current implementation includes a set of socket-based servers that gather and organize monitoring data based on symbolic expressions (s-exps). Its architecture is hierarchical, that is, each node is treated as a leaf node in a tree while a series of data concentrators gather data from the nodes and transport it to a root. The root then provides data to clients.

The Supermon system consists of several components. A. Within the compute-node OS is a kernel-module that exports system-level parameters, locally, as a file in a sysmon pseudo-filesystem. B. On each compute node is the mon daemon. It reads the file under /sysmon and makes available the system-level metrics as s-exps over the network via TCP/IP. C. Mon also listens on a Unix Domain Socket locally accepting data from any other sources. This interface to mon is the monhole. D. On the server side, there is the supermon daemon. This talks to all the monhols on the compute nodes and gathers and collects the data from them. This data includes the system-level parameters as well as data submitted through the monhole interface. Supermon then makes this data available as another s-expression to any interested clients.

Adapting Supermon and TAU

Supermon: The Mon daemon exports the Unix Domain Socket based monhole interface for external sources to inject data. Due the monhole not having been exercised much it had to be tested and bug-fixed before TAU could use it. Next its buffering characteristics needed to be made more flexible. Only a simple and efficient ‘replace-read’ policy was available. This had several advantages: i) slow sinks will not cause infinitely large buffers to be maintained at the mon daemon and ii) multiple sinks can query the data simultaneously without race conditions. But its cons include the problem of data-loss for even transient slowdowns of sinks and bandwidth wastage when sink query rate becomes greater than data generation rate. Even a small configurable buffer can alleviate the former, whereas a ‘replace-draw’ strategy when using a single client can remedy the latter. A runtime-configurable buffer strategy was put in place.

The repair mechanism for hierarchical topologies was also fixed.

Experiment Configurations

We compare the performance of the NAS4/Parallel LU (Class B) application benchmark under different configurations, including: 1. vanilla-uninstrumented LU. 2. LU instrumented with TAU to generate post-mortem measurement data. 3. LU instrumented with TAU for online measurement using NFS as the transport and 4. Online TAU instrumented-LU measurement over Supermon transport. Online measurement is performed at a frequency of 1 Hz. We repeat each of the runs over 16, 32 and 64 nodes to examine scalability. The China City cluster from Argonne National Lab, with Pentium III dual-CPU SMP Linux nodes running Ethernet, serves as our test environment.

Performance and Scalability

 Increasing Scalability Further

We outline three methods that are works-in-progress to further improve the scalability of this system. The methods either try to reduce the number of nodes touched per-query or try to reduce the data generated per node per query.

1. Optimal Schedule

The entire suite of data provided by the cluster includes values that vary over time at different scales. For example, values such as temperature, fan behavior, and various system power voltages vary at a much coarser grain than CPU utilization. If a user wishes to look at both values, it is wasteful to repeatedly sample the coarse grain variables that change slowly over time while gathering other data. Therefore, by applying a priori knowledge of the variables to sample and the rates at which they are relevantly changing, one can derive a sampling schedule to gather exactly what data is changing at any given time. For TAU data, only the measurements of those routines (events) that have changed since the last query will be returned.

2. Sampling Contexts

Instead of querying every node at every interval, we can sample the cluster-address space and the application namespace the MPI Ranks). When such a ‘sampled’ query goes out to the mon-daemons, they can each first ask a ‘sampler’ if this query should be satisfied before replying. A default sampler may reply TRUE with probability \(k(1/N)\) where N is the number of contexts (e.g. MPI Ranks) and \(k\)=const is a configurable constant. In practice, the number of nodes in the system, this will bound, on average, the number of nodes touched per query. More sophisticated sampling techniques can be applied.

3. Data Aggregation with Hierarchical Topology

Determining the transport hierarchy (i.e. a tree) can improve scalability. The leaves would be the actual mon daemons (and the application contexts). But sending data loss-less over the tree will simply lead to the total amount of data being magnified (every internal node retransmits data again) - something we want to avoid. Therefore some type of ‘aggregation’ is required to go along with the hierarchical structure. The aggregation strategy may depend on the level (or depth) of the internal node in the tree. A deep internal node may find the average (or max, sum) of the data from its children, but higher level internal-nodes may resort to loss-less aggregation.

Conclusion

We have obtained recent work in creating a large-scale online application performance monitor by using Supermon as the underlying transport for the TAU system. The rationale behind our approach and the architecture of the system is both explained. Our experimental results suggest that this strategy is far better than using a traditional NFS as the transport both in terms of performance and scalability. We have also listed several strategies we are experimenting with to improve scalability further.

Future Work

We would like to pursue the below efforts in future: 1. Implement and evaluate strategies listed to improve scalability further. 2. Experiment on very large scale platforms such as BGL (already portable). 3. Implement measurement control using Supermon as a reverse channel from sink to application. 4. Add new custom transports to TAU such as MRNet from University of Wisconsin. 5. Correlate application performance with system-level metrics (e.g. sensors).

Performance Results

The graphs clearly highlight the following:

1. Overhead of obtaining post-mortem TAU measurements < 6%.
2. TAU overhead grows logarithmically over number of CPUs.
3. Overhead of online performance measurement/data-retrieval using NFS grows super-linearly and is as high as 70%.
4. Overhead of online performance measurement/data-retrieval using Supermon also shows logarithmic growth. It is also very close to the TAU overhead, hence provides online measurement almost for free.
5. As CPU count grows, the savings obtained from Supermon transport over NFS grows super-linearly.