High Fidelity Data Collection and Transport Service Applied to the Cray XE6/XK6

Jim Brandt*, Tom Tucker†, Ann Gentile*, David Thompson§, Victor Kuhns‡, and Jason Repik‡

*Sandia National Laboratories, Scientific Computing Systems, Albuquerque, NM
† Open Grid Computing, Austin, TX
§ Kitware Inc., Carrboro, NC
‡ Cray Inc., Albuquerque, NM

*Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.
Outline

• Motivation
• High-level Overview of Data Collection and Transport
• Case Study: Resource-Aware Application
• Enhancements
• Overhead
• Summary & Future Work
Motivation

Gain insight into resource utilization/bottlenecks (e.g. network bandwidth/hotspots, CPU utilization, Memory footprint/bandwidth)

- Intelligent job placement
- Run-time workload partitioning/adaptation
- Historical comparison
- Anomaly detection
Monitoring System and Application Resource Utilization

- Typical monitoring systems target failure detection, uptime, and resource state/trend overview:
  - Information targeted to system administration
  - Collection intervals of minutes
  - Relatively high overhead (both compute node and aggregators)

- Application profiling/debugging/tracing tools:
  - Collection intervals of sub-seconds (even sub-millisecond)
  - Typically requires linking, not run under real-world conditions (i.e. tools perturb the application profile)
  - Limits on scale
  - Don’t account for external applications competing for the same resource

- Lightweight Distributed Metric Service (LDMS):
  - Continuous data collection, transport, storage as a system service
  - Targets system administrators, users, and applications
  - Enables collection of a reasonably large number of metrics with collection periods that enable job-centric resource utilization analysis and run-time anomaly detection
  - Variable collection period (~seconds)
  - On-node interface to run-time data
LDMS High Level Overview

- Only the current data is retained on-node
LDMS Functional Overview

- Data is bundled into “Metric Sets” – this is the granularity of storage and query
- Metric Sets have associated Data and Meta-data and include generation numbers for both
  - Meta-data is only transmitted during initial setup and when change occurs
- Run-time plugin add, start, stop
  - Add new collection components
  - Start collection – begin scheduling data collection and make data visible to queries
  - Stop collection – stop scheduling data collection, last data set still visible to queries – no CPU overhead associated with this as no collection scheduled
  - Modify collection frequency – change the length of time between collection on a per data set basis
- Queries can be either host local or remote
- Socket or RDMA transport options
**LDMS Dataset Example**

- `shuttle-cray.ran.sandia.gov_1/meminfo`
  - U64 1 `component_id`
  - U64 160032 `MemFree`
  - U64 181728 `Buffers`
  - U64 3443332 `Cached`
  - U64 33076 `SwapCached`
  - U64 2987544 `Active`

- `shuttle-cray.ran.sandia.gov_1/procstatutil`
  - U64 1 `component_id`
  - U64 1826564 `cpu0_user_raw`
  - U64 699631 `cpu0_sys_raw`
  - U64 663843760 `cpu0_idle_raw`
  - U64 201018 `cpu0_iowait_raw`

- `shuttle-cray.ran.sandia.gov_1/vmstat`
  - U64 1 `component_id`
  - U64 40008 `nr_free_pages`
  - U64 122286 `nr_inactive_anon`
  - U64 321902 `nr_active_anon`
  - U64 465532 `nr_inactive_file`
  - U64 424986 `nr_active_file`

- **Metric sets:**
  - (datatype, value, metricname) tuples
  - Associated with a unique `component_id`

- **API:**
  - `ldms_get_set`,
  - `ldms_get_metric`  
    `ldms_get_u64`

- Same API for on-node and off-node (aggregator) transport
Metric Set Format

- Meta data generation number bumped whenever metrics are added or removed
- Data generation number changes whenever a value changes
- Meta data generation number is included with metric data to detect when cached local metadata is stale
Current Data Collectors/Storage

- /proc
  - meminfo, vmstat, stat, interrupts, pid/(stat, statm)
  - Kgnilnd (Cray specific)
- gemctrs (Cray specific)
  - Gemini Tile and NIC counters
- nicctrs (Cray specific)
  - Gemini NIC counters
- perf_event
  - Generic interface for acquisition of hardware counters e.g. data cache misses, instruction cache misses, hyper-transport bandwidth (AMD)
- rsyslog (Cray specific)
  - SEDC (RAS) and ALPS data
- Lmsensors (/sys)
  - Temperatures, fan speeds, voltages
- Flat File, MySQL, CSV, Custom
Case Study: Resource-Aware Application

- Performance of an application depends on capabilities of the hardware and system software resources and on how the application utilizes them.
- Assess the viability of enabling distributed HPC applications to utilize node level monitoring information to make run-time load balancing decisions.

- SIERRA Applications repartition using Zoltan
- Augment Zoltan to acquire and utilize data from LDMS
- Utilize remote analysis of data to determine metrics of interest
Architecture: LDMS on Cielo

Cielo (Cray XE6)
compute nodes:
subset: 646 nodes, 10336 cores

Cielo
viz nodes

Whitney
132

Whitney
admin nodes

database/file
database/file

Aria
modified Zoltan

Idmsd

DISCOM

Idmsd w/ data collection
• Small scale dynamic application
• Particle transport results in load imbalance in changing partition size and location
• Include run-time CPU utilization in partitioning calculation
• Improvement over all processors (but well-balanced to begin with)
Aria on Cielo: 10112 processors

- Thermal code, generally well balanced
- Off-node post-processing analysis to determine variables of interest
- Processors 0 and 9 exhibit more non-voluntary context switches and interrupts and are assigned smaller partition sizes
Non-Voluntary Context Switches

- From 8310 processor application run on Cray XE6
Aria on CDS: 8310 processors

- Processors with higher idle/user cycles are assigned larger partition sizes
- Self-correcting repartitioning: distribution tightens after initial over-correction for idle cycles
Enhancements

• *ldmsd* plugin interface for collectors and storage
  – Single daemon

• Implemented RDMA over Gemini transport
  – CLE4.0 UP03 – Enabled allocation of System pTag
- CPU overhead increases with number of metrics in a metric set for a particular gathering mechanism (e.g. /proc readers, /sys readers, ioctl calls)
- gemctrs – ioctl as opposed to reading from /proc. gemctrs with ~300 metrics has < 1/3rd the overhead of procstatutil but has ~twice the number of metrics
Overhead Summary

CPU Overhead
- Mostly due to data collection vs. transport
- RDMA (UGNI) has none past startup
- SOCK significant if collection overhead is small (e.g. small dataset)

Memory Footprint
- RDMA has a larger memory footprint than SOCK
- Except for gemctrs, sampler overhead, over ldmsd alone, is about the same
Summary

• Lightweight Distributed Metric Service:
  – System service that provides low-overhead remote storage of and on-node access to high-fidelity system related data

• Demonstrated viability for use in analysis and runtime repartitioning of production HPC applications on XE6

• Lowest overhead: efficiently gather small set of targeted data of interest and use RDMA for transport

• Adding collector plugins doesn’t substantially increase memory footprint (e.g. only increase is data + metadata + accounting)
Future Work

• Investigate perturbation to large scale applications
  – Priority
  – Kernel collection modules
  – Metric set size

• Presentation of LDMS data in architectural context:
  – Inter-node congestion
  – Intra-node memory bandwidth sharing
Questions?