Studying System Support for a Key Value Store

Boston University SESA Group
Dan Schatzberg
Outline

0. Context
1. High-level architecture
2. Software decomposition
3. Key-value store support

Exploring new System Software Models for future Data Center Scale Systems
Reusable Component Runtime for Constructing Scalable Elastic Services
System Support for Key-Value Stores
Research Background

• Datacenter scale systems are of increasing importance

• Scale-out applications not just in HPC but also in cloud environments

• Increasing complexity

• Heterogeneity

• Failures/Elasticity
High-level Architecture

- Doing away with the kernel-userspace boundary
- Software is constructed as libraries on top of a thin scalable runtime
- Low level primitives to aide in the construction of distributed software
- Allow incremental porting of legacy software
High-level Architecture

Application

Libraries

Scalable Runtime
Scalable Runtime
Scalable Runtime
Scalable Runtime

Legacy OS
Component Level Decomposition

- Managing communication and locality is hard
- Inspired by work on Tornado/K42 - no communication paradigm is best
  - Not only message passing
  - Not only shared memory or RDMA
- Encapsulate communication by decomposing software as components
Elastic Building Blocks

- Invocation of an object goes through a translation table
- Potentially different representatives of an object per processor
Elastic Building Blocks

- Objects respond to events
  - EbbCall
  - First time accessed on a processor
- Also hardware “events”
  - Tree packet arrived
  - Timer interrupt fired
Component Level Decomposition
Our Goals

• Explore utility of system level primitives for Key-Value Stores (KVSs):
  • Can they help with faults / elasticity?
  • Can they encapsulate HW level optimization?
  • Can they help achieve HPC scale KVSs?
Status

- “Have” EbbOS runtimes for Linux, x86_64, PPC32, PPC64
- Gathering baseline measurements of event dispatch costs for a USENIX poster
- Developing an Ebb that implements a hash table
Backup
Clustered Object Instance

Global Translation Table (table of Root pointers)

Local Translation Tables (tables of Rep pointers)

\[ \text{COID} = \text{vbase} + (i \times \text{sizeof(table entry)}) \]
Background
Reliability is a key problem at scale: machines fail. Here, node reliability is two orders of magnitude higher than a commodity server.

Blue Gene's control network provides access to all parts of the hardware outages so that the nodes, for example, reconfigure themselves. Thus, even the initial firmware which initializes nodes contained from 16 up to all nodes in an installation.

Failure is considered a machine failure. Hence, individual node reliability is guaranteed. While in many cases we may use a forwarding node at the physical layer, a network segment may become unreachable. While we can easily deallocate a node, the node is reset, the memory gets scrubbed, and the node is placed back into the pool ready to be handed out again.

Upon each node allocation we return a list of node addresses in the node card with 32 compute nodes and 2 IO nodes (two 10G Ethernet connectors on the front). The boot loader which is placed on all nodes by the existing firmware (see also Section 4.5) which allows interaction with the hardware outages so that the nodes, for example, reconfigure themselves. Thus, even the initial firmware which initializes nodes contains from 16 up to all nodes in an installation.

Blue Gene's original control system was constructed for a reliability target of 7 days mean time between failure for a machine with 72 racks, the hardware is loaded via the control network into each node card form a rack. A forwarding node at the physical layer, a network segment may become unreachable. While we can easily deallocate a node, the node is reset, the memory gets scrubbed, and the node is placed back into the pool ready to be handed out again. Upon each node allocation we return a list of node addresses in the node card with 32 compute nodes and 2 IO nodes (two 10G Ethernet connectors on the front). The boot loader which is placed on all nodes by the existing firmware (see also Section 4.5) which allows interaction with the hardware outages so that the nodes, for example, reconfigure themselves. Thus, even the initial firmware which initializes nodes contains from 16 up to all nodes in an installation.

Blue Gene's original control system was constructed for a reliability target of 7 days mean time between failure for a machine with 72 racks. The hardware is loaded via the control network into each node card form a rack. A forwarding node at the physical layer, a network segment may become unreachable. While we can easily deallocate a node, the node is reset, the memory gets scrubbed, and the node is placed back into the pool ready to be handed out again. Upon each node allocation we return a list of node addresses in the node card with 32 compute nodes and 2 IO nodes (two 10G Ethernet connectors on the front). The boot loader which is placed on all nodes by the existing firmware (see also Section 4.5) which allows interaction with the hardware outages so that the nodes, for example, reconfigure themselves. Thus, even the initial firmware which initializes nodes contains from 16 up to all nodes in an installation.

Blue Gene's control network provides access to all parts of the hardware outages so that the nodes, for example, reconfigure themselves. Thus, even the initial firmware which initializes nodes contains from 16 up to all nodes in an installation.
Research Background

• We have been making large scale applications by building collections of single node operating systems stitched together with middleware

• Applications are single user, multi node
High-level Architecture

Applications

Legacy OS
High-level Architecture

Application

Middleware
Legacy OS

Middleware
Legacy OS

Middleware
Legacy OS

Middleware
Legacy OS
High-level Architecture