Fault Oblivious Extreme Scale Execution Environment (FOX)

LLNL: Maya Gokhale, Roger Pearce, Scott Lloyd

SNL: John Floren, Jeremy Wilke

PNNL: Sriram Krishnamoorthy, Andres Marquez

IBM: Evan Speight

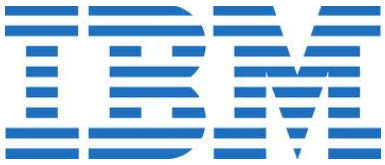
FOX Team

Boston University: Jonathan Appavoo, Dan Schatzberg

Ohio State: Saday Sadayappan

Bell Labs: Noah Evans, Jim McKie

Alumni:
Ron Minnich & Curt Janssen, Google
Eric Van Hensbergen, ARM
Jimi Xenidis, Qualcom

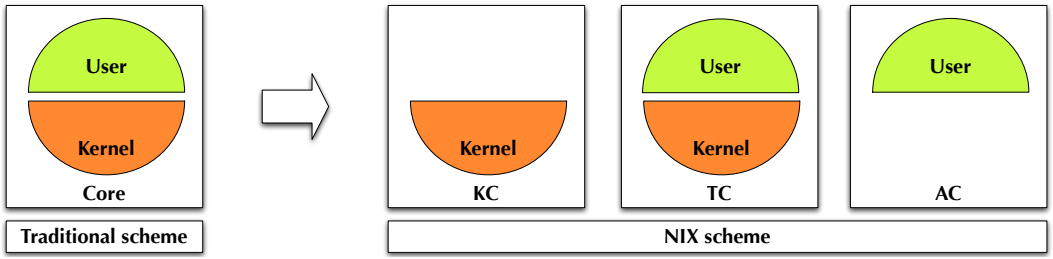


Outline

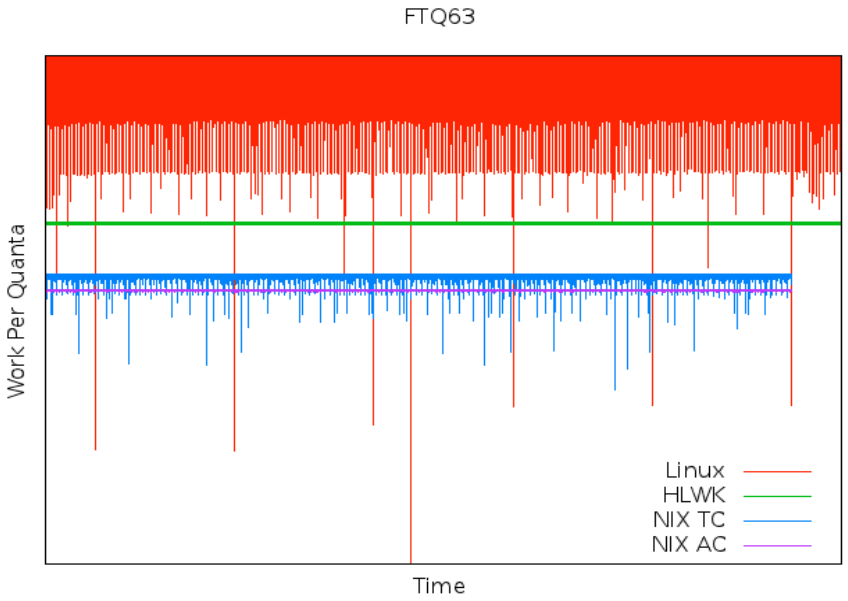
- OS innovations
  - Many core
  - Bare metal application execution
  - Support for large memory
  - Considerations for legacy
- Task oriented programming models
  - Reliable data substrate
  - Alternative models
- Graph application framework
  - Asynchronous, massively concurrent
  - Transparent access to memory hierarchy encompassing DRAM/NVRAM

NxM OS for multi-core

- Build an operating system specifically for new heterogeneous manycore CPUs in future exascale systems
- Cores now have *roles*
  - Standard Timesharing Cores (TC): a common core running kernel and user code in a time sharing fashion
  - Dedicated application cores (AC): a core running user code without any interrupts (even without clock interrupts)
  - Kernel cores (KC): a core that only runs kernel code on demand
  - Cores communicate by sending active messages that include a function to be executed and its arguments
  - Programs transparently change roles by switching cores
- Two page sizes: 2MB and 1GB
  - 64GB – 63 1GB pages and 512 2MB pages instead of 16M 4K pages
- Emulates many Linux system calls
  - App can use both NxM and Linux system calls
- Demonstrated miniFE performance equal to native Linux
- Demonstrated iSCSI user level driver



Bell Labs, Sandia



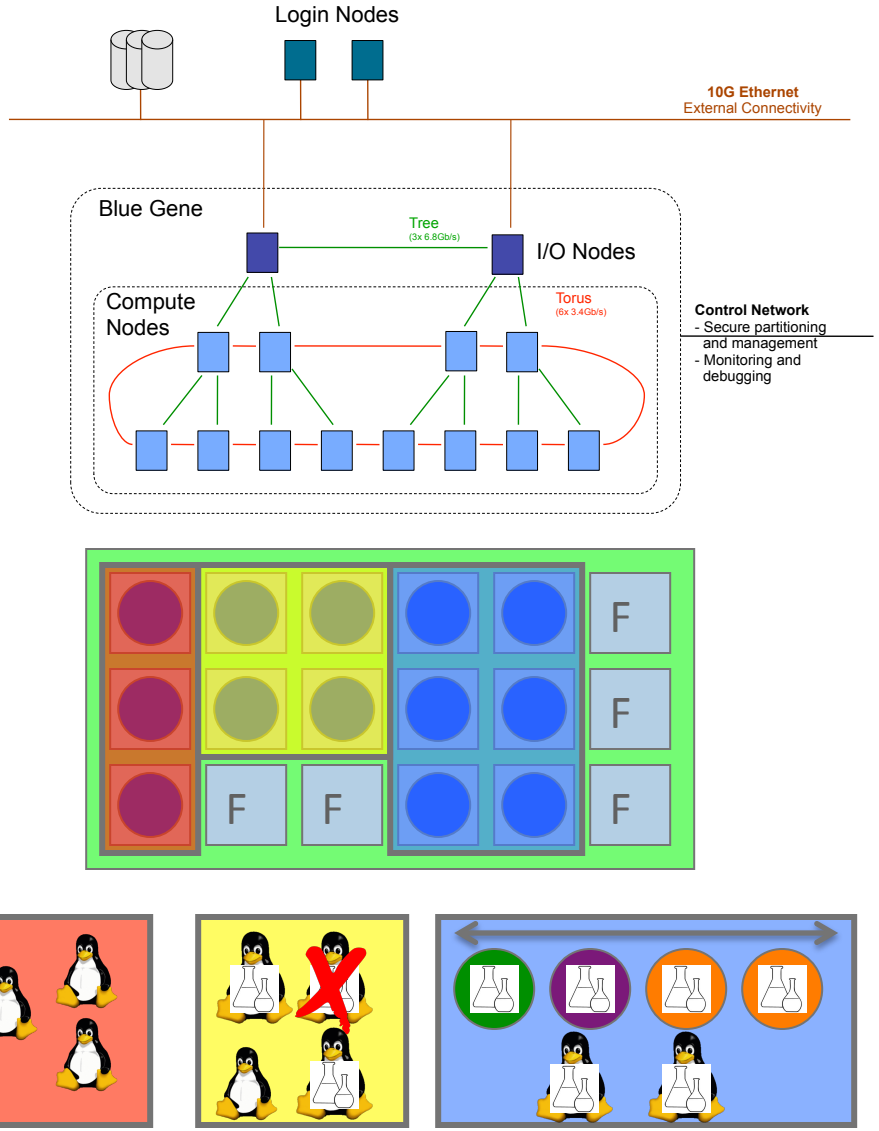
IBM Research

Global Collaboration: Bell Labs, Sandia CA, Rey Juan Carlos U., Google
Website: http://nxm.coreboot.org/NxM
Source code: http://nxm.coreboot.org/Get_NxM

Kittyhawk BG/P infrastructure

Enables dynamic heterogeneous computing on BlueGene Systems

- Applications can be composed of arbitrary mixes of compute nodes running diverse software stacks
- Allow compute nodes to run application specific system software down to the granularity of individual nodes.
- Application can run in a customized environment with specific OS, OS rev., libraries
- Nodes can be dynamically allocated and de-allocated
- Modified kernels make it possible to simulate node level faults
- Convenient environment for exploring alternative system software and configurations



Jonathan Appavoo, BU

Boston University

Website: http://kittyhawk.bu.edu/kittyhawk/Kittyhawk.html

Source code: http://git.anl-external.org/kittyhawk/

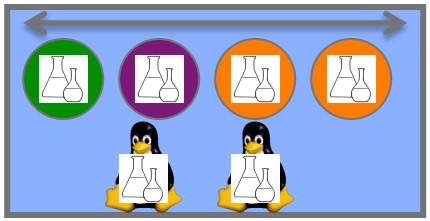
Scalable and Elastic System Software for FOX

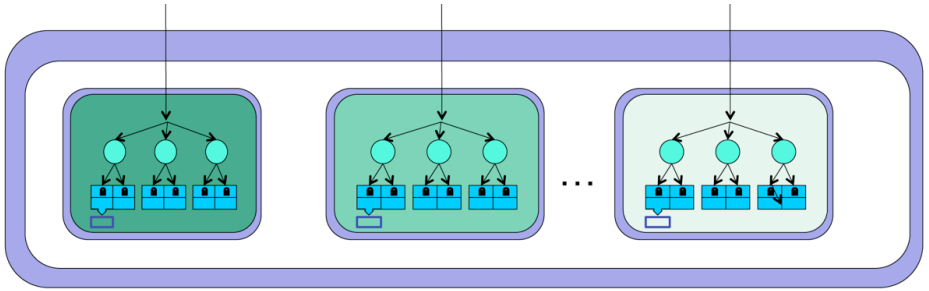
Explore how to support scale and and elasticity in system software

- Assumes dynamic heterogeneous environment where nodes can come and go
- Software can be composed of mix of commodity and custom code
- Support the development of new libraries of "Elastic building block" software that encapsulates function such as hash tables that are customized for specific application access patterns and underlying hardware communication facilities
- Facilitate incremental development by providing both "bare-metal" execution and traditional OS integration
- Target is a focused trial of the ideas in the context of an alternative implementation of a FOX project key-value store application.

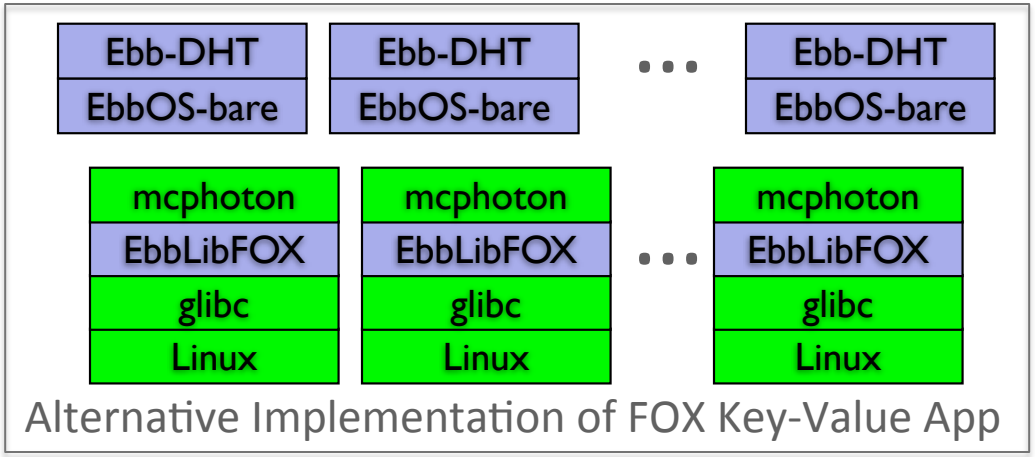
Jonathan Appavoo, BU

System Software designed to support Elastic High Performance Software





Elastic Building Block DHT (Ebb-DHT)



Alternative Implementation of FOX Key-Value App

Task-oriented programming libraries

Libraries support

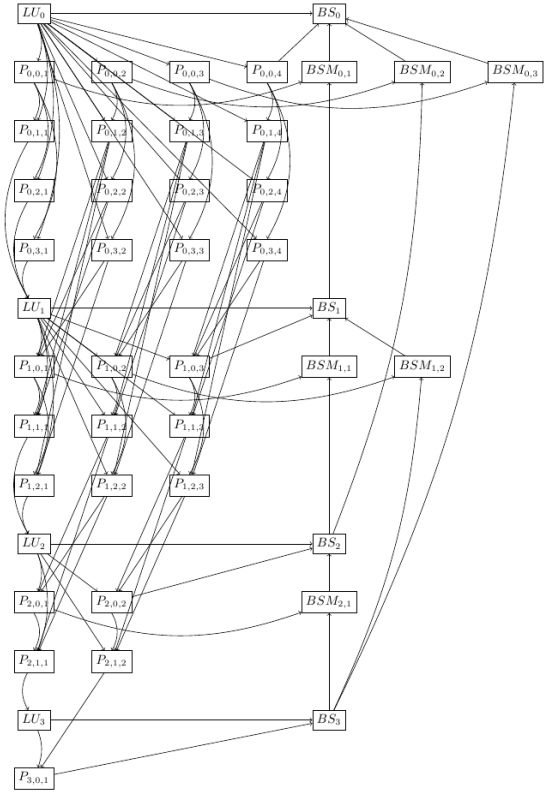
- Lightweight tasks
- Asynchronous task model
  - Data dependence among tasks
  - One-sided communication
  - Varying duration computational tasks: load balance challenge
- Underlying reliable data store

$$P_2 \begin{bmatrix} LT_2 \\ L_{2,0} \end{bmatrix} UT_2 \leftarrow \text{factor} \begin{bmatrix} A_{2,0,0} \\ A_{2,1,0} \end{bmatrix}$$

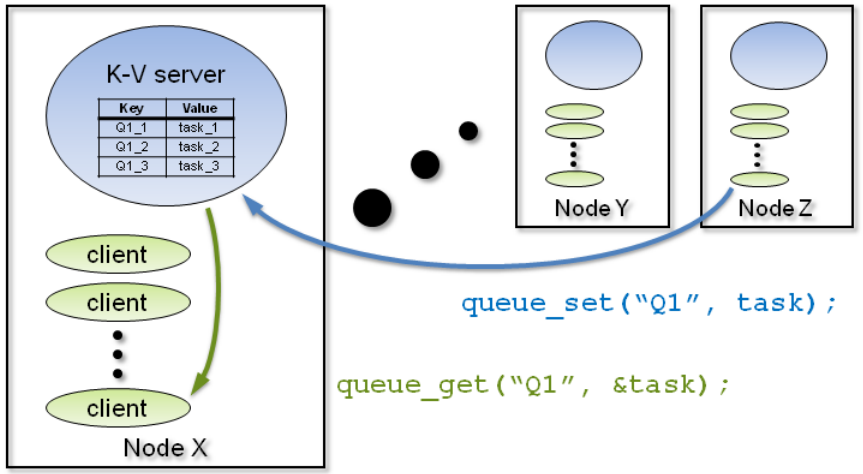
$$\begin{bmatrix} A'_{2,0,0} \\ A'_{2,1,0} \end{bmatrix} \leftarrow P_2 \begin{bmatrix} A_{2,0,0} \\ A_{2,1,0} \end{bmatrix}$$

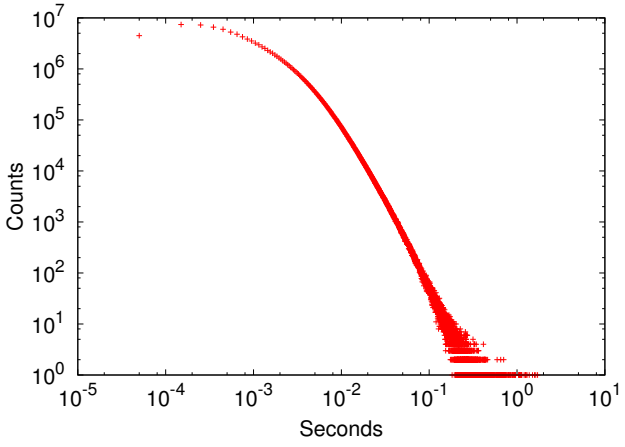
$$U_{2,0} \leftarrow \text{solve } LT_2 U_{2,0} = A'_{2,0,0}$$

$$A_{3,0,0} \leftarrow A'_{2,1,1} - L_{2,0} U_{2,0}$$



Work queue in K-V store





Reliable data store

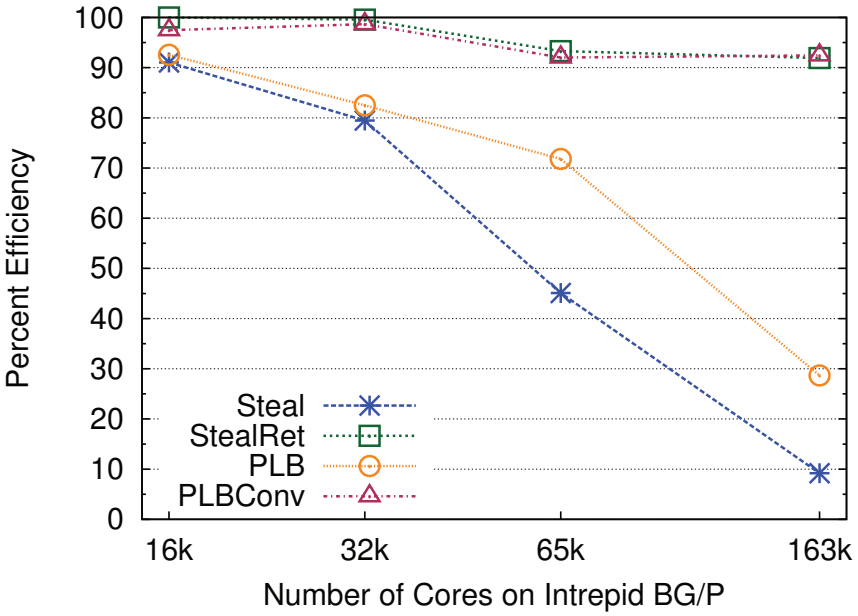
- Approach 1 (conservative): local persistent memory for checkpoints
- Approach 2: Global Arrays + MPI: task model within a major iteration step of the simulation
  - Peer to peer, load balance with work stealing
- Approach 3: Reliable key/value store to hold task queues and possibly data
  - Hierarchical – client/server
- Approach 4: fault tolerant tuple space

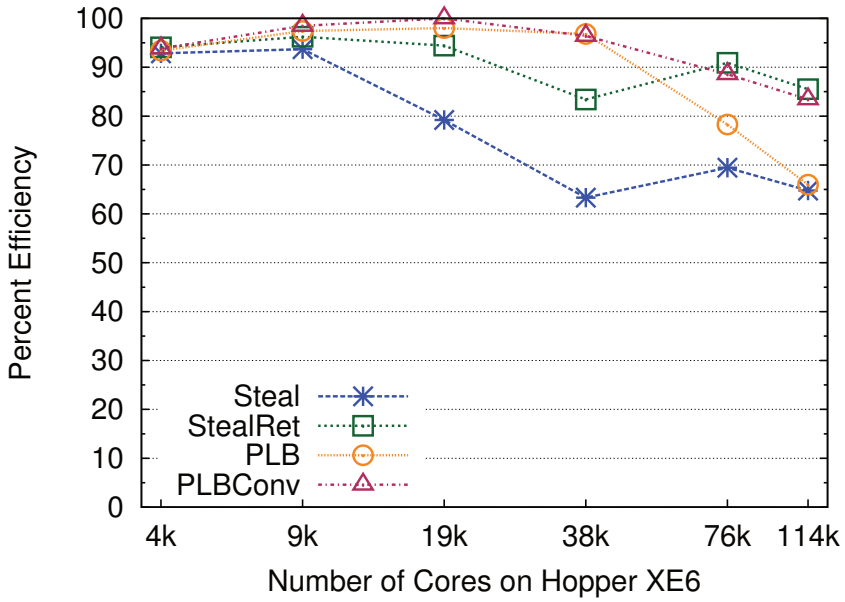
Approach 1: Checkpoint in NVRAM

- A slight twist to checkpointing
- Maintain a persistent heap in local NVRAM
  - Mmap a file to back the heap
  - Mark global checkpoint data in source program with attribute PERM
  - Allocate dynamic checkpoint data with perm allocator
    - Allocates in the persistent heap
  - Perm library msyncs file to do a checkpoint
  - To restore a checkpoint, mmap file back in at previous virtual address
- Integrated into a couple of codes
  - LULESH, ParaDys, LAMPPS

```
 struct Domain *domain = pinit ? (pdom = PERM_NEW(Domain)) : pdom ;
…
 if (pinit) domain->e = PERM_NEW(Real_t[domElems]) ;   /* energy */
 if (pinit) domain->p = PERM_NEW(Real_t[domElems]) ;   /* pressure */
…
/* Persistent memory initialization */
   perm(&pdom, sizeof(pdom));
   mopen(mpath, "w+", MMAP_SIZE);
   free(mpath);
   PermRestart(&pinit);
```

Approach 2: Task-based Model with Global Arrays





Dynamic load balancing can smooth out variations

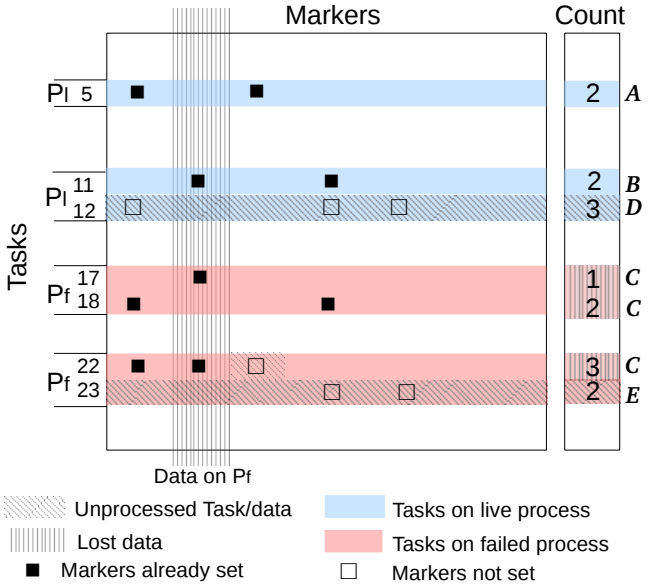
- Irregular computation
- Faults
- System noise
- Energy constraints

Distributed algorithm

- Retention based work stealing
  - Stolen tasks are retained in next iteration
- Profile-based load balancing
- Scales well with these optimizations

Sriram Krishnamoorthy, PNNL
Saday Sadayappan, OSU

Approach 2: Fault tolerance

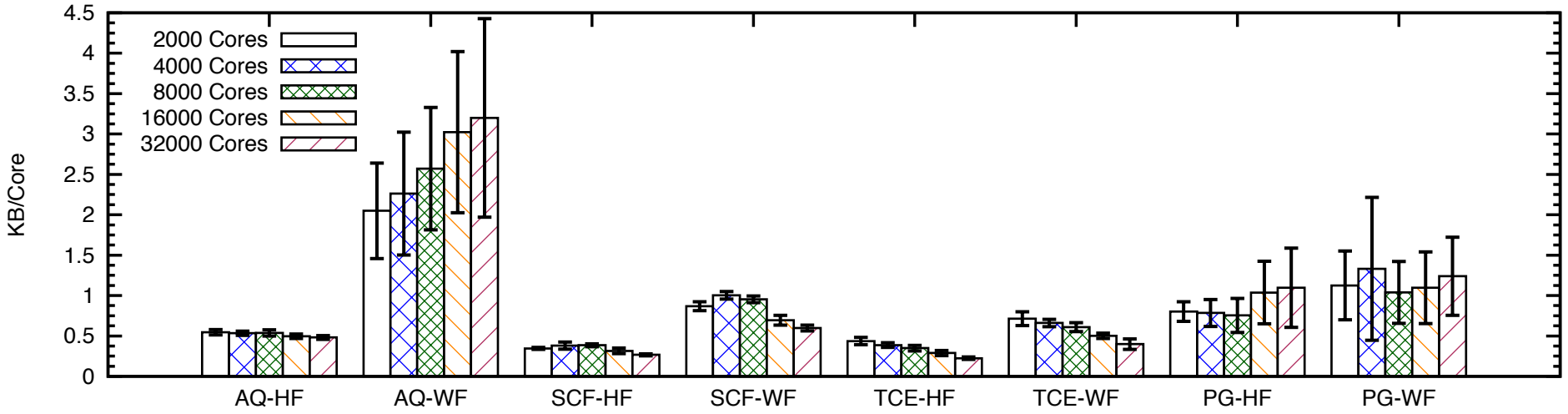


We have developed fault tolerance mechanisms for task parallel computations employing work stealing, without requiring frequent synchronizations, collective roll back, or message logging. The completion of data operations is tracked using markers to maintain a consistent data store.

This information is used to accurately classify tasks as: (A) executed by a live process with no data loss; (B) executed by a live process with data loss; (C) executed by a failed process; (D) enqueued on a live process; or (E) enqueued on a failed process. This classification is used to determine the tasks to be re-executed. We demonstrated that the overheads (space and time) of the fault tolerance mechanism are low, the cost incurred due to failures are small, and the overheads decrease with per-process work at scale.

Sriram Krishnamoorthy, PNNL
Saday Sadayappan, OSU

Approach 2: Tracing work stealing schedulers

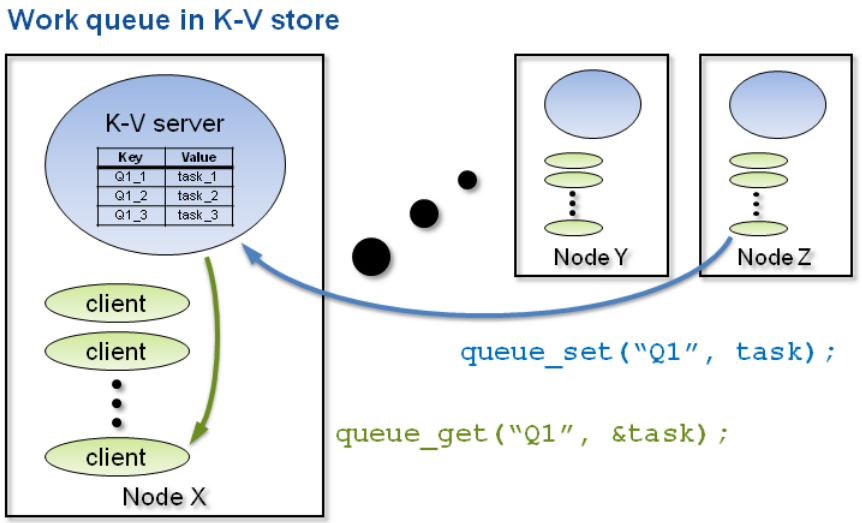


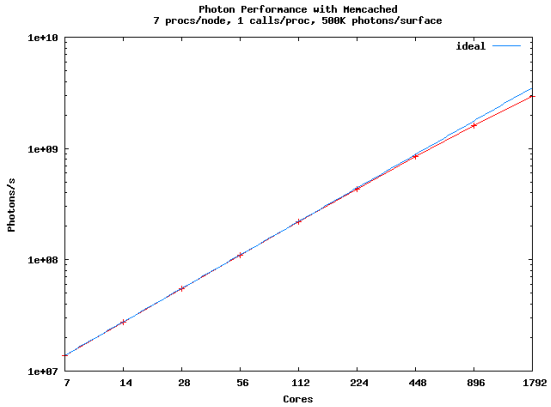
We have developed algorithms to trace async-finish programs scheduled using work-first (WF) and help-first (HF) work stealing schedulers; The algorithms exploit the stealing relationship to derive compact traces, shown above for AllQueens(AQ), SCF, TCE, and pgraph (PG) benchmarks on OLCF Titan; these algorithms were then used to enable retentive work stealing for recursive parallel programs and optimize data race detection for async-finish programs.

Sriram Krishnamoorthy, PNNL

Approach 3: task queues in key-value store

- Adapted reliable, distributed key-value developed in commercial sector to HPC to hold distributed work queues
- Current version uses Memcached from Couchbase or Kyoto Tycoon
  - No MPI!
  - Fault handling server
- Developed prototype application programming interface (API) called libfox
  - Represent work in key-value store paradigm
  - Broadcast parameter set to workers
  - Distribute tasks to workers
  - Collect results from distributed data store
- Task queue performance on 2K cores
  - Queue set: 626 usec
  - Queue get: 450 usec
- Demonstrated on LLNL Hyperion cluster
  - (2K cores)
  - task parallel benchmark
  - Linear scaling tails off due to global operations



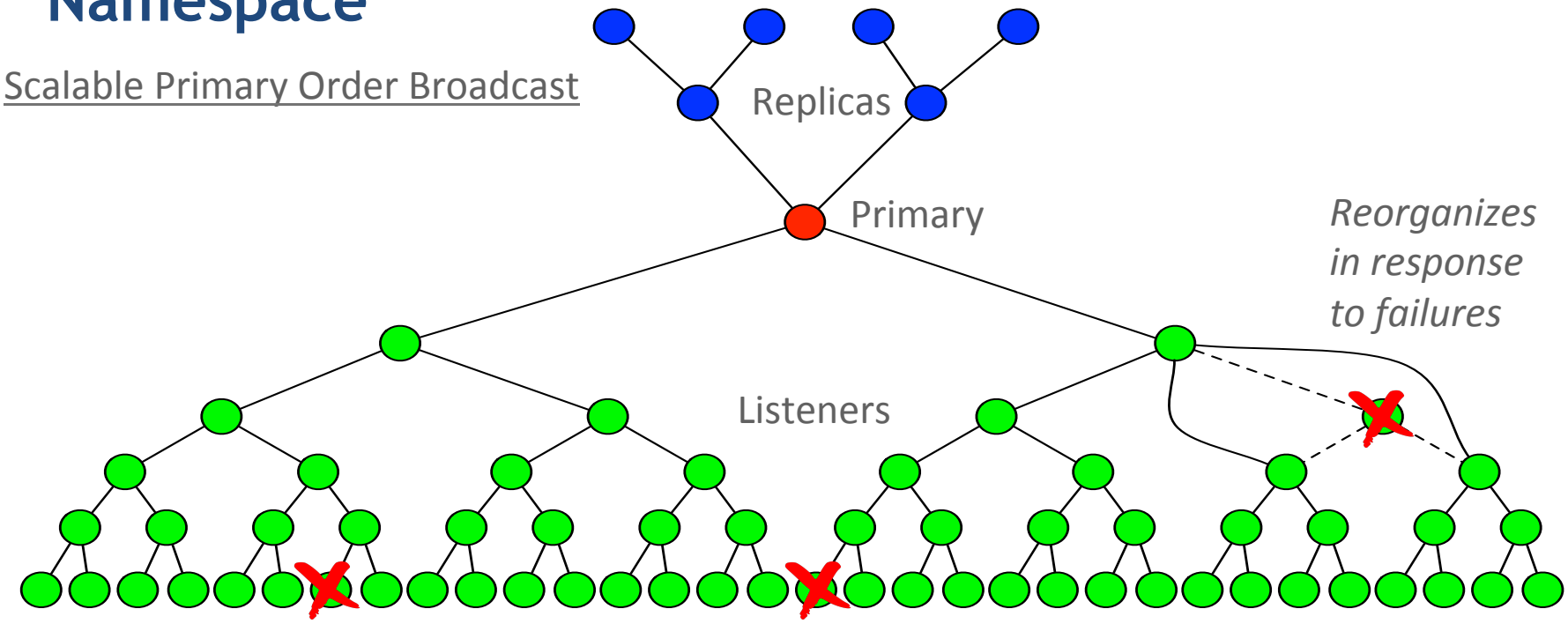


Scott Lloyd, LLNL

- Key/value store is being implemented on BG/P to support global ID name space for Elastic building blocks by BU

Algorithm for System Provided Strongly-Consistent Namespace

Scalable Primary Order Broadcast



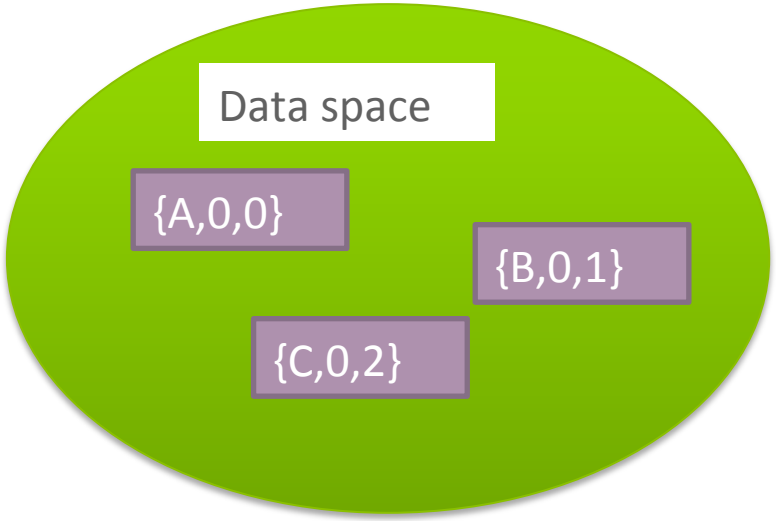
- Updates of data is ordered by the Primary
- Replicas ensure failure of Primary can be tolerated
- Functioning Listeners are guaranteed to observe data updates in order
- All reads are local – High Performance
- Writes go through the Primary and have logarithmic latency
- Structure is a self-organizing overlay that can be efficiently mapped to hardware topologies
- Listener failure does not delay global progress

Approach 4: Tuple spaces … in progress

Jeremy Wilke, Sandia CA

I. Influenced by Linda, Concurrent Collections (CnC), Charm++, DaGuE.
II. Resilience by ensuring tuple space put/get actions are fault-tolerant transactions.
III. Tuple transactions must be ‘’staged’’ or ‘’mirrored’’ to some temporary area in case of failure
IV. All functionality is built on top of tuple space transactions. Resilience strategy can maintain narrow focus on (relatively) simple abstraction of tuple space.

What’s different from other tuple space methods?



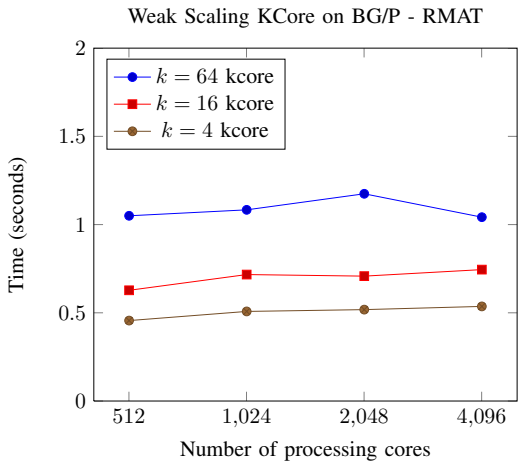
1) Hybrid of Linda/CnC. Tasks can be requested as wildcard tuple Task(0,int?,int?) with *actuals* (specific value) and *formals* (any value)
2) Formals give flexibility. Workers don’t have to run specific task (decoupled in space and time)
3) Actuals can give hints to runtime about locality (which node *should probably* run a task)
4) ARBITRARY event handlers/listeners can be attached to tuple puts/gets

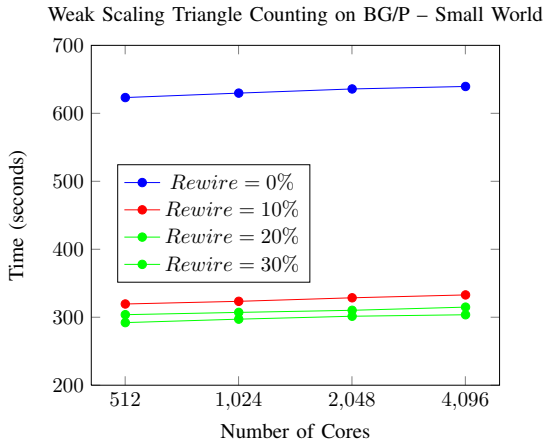
Parallel graph traversal framework

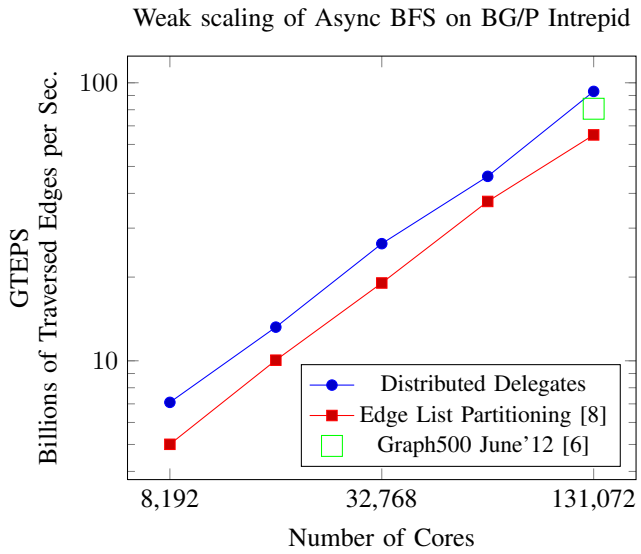
- High visibility data intensive supercomputing application class
- Traverse scale free graph with trillions of edges
  - Social network, web
- Developed parallel, latency tolerant asynchronous traversal framework
- Scalable: single server with NVRAM to data intensive cluster to BG/P
- Uses visitor abstraction
  - Visitor is an application-specific kernel that the framework applies to each graph vertex
  - Visit results in traversal of graph edges to queue work on target vertices
  - Visitor is queued to the vertex using priority queue
- Demonstrated with
  - Breadth first search
  - Single source shortest path
  - (Strongly) Connected components
  - Triangle counting
  - K-th core
- Multiple Graph500 submissions
- Code is being open sourced

Roger Pearce, LLNL & TAMU
Maya Gokhale, LLNL
Nancy Amato, TAMU

Graph scaling results







Conclusions, credits

- Fox advanced new concepts in core specialized OS
  - NxM
  - Supported IBM FusedOS
  - Kittyhawk
  - Elastic Building Blocks
- We developed task libraries designed for irregular, dynamic parallelism that would be resilient in the presence of node failure
  - Global Array based Tascel
  - K/V store based libfox
  - Tuple space library in progress
- We developed latency tolerant framework for graph traversal
- **Thanks to ALCF INCITE program, NERSC, OLCF, LLNL**
- Code
  - http://nxm.coreboot.org/Get_NxM
  - http://git.anl-external.org/kittyhawk
  - https://computation.llnl.gov/casc/dcca-pub/dcca/Downloads_files/perm-je-latest.tar.gz

Papers

- Jan Stoess, Udo Steinberg, Volkmar Uhlig, Jonathan Appavoo, Amos Waterland, Jens Kehne, Kittyhawk. A lightweight virtual machine monitor for Blue Gene/P, International Journal of High Performance Computing Applications, March 27, 2012
- Francisco J. Ballesteros, Noah Evans, Charles Forsyth, Gorka Guardiola, Jim McKie, Ron Minnich, Enrique Soriano-Salvador, NIX: a case for a manycore system for cloud computing, Bell Labs Technical Journal, 2012
- J. Lifflander, S. Krishnamoorthy, and L. V. Kale, "Work stealing and persistence-based load balancers for iterative overdecomposed applications," in Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing, pp. 137–148, ACM, 2012.
- J. Daily, S. Krishnamoorthy, and A. Kalyanaraman, "Towards scalable optimal sequence homology detection," in Workshop on Parallel Algorithms and Software for Analysis of Massive Graphs (ParGraph), 2012.
- J. Lifflander, S. Krishnamoorthy, and L. V. Kale, "Steal tree: Low-overhead tracing of work stealing schedulers," in PLDI, ACM, 2013.
- H. Arafat, J. Dinan, S. Krishnamoorthy, P. Balaji, , and P. Sadayappan, "Work stealing for GPU- accelerated parallel programs in a global address space framework," Concurrency and Computation: Practice and Experience special issue on "Productive Programming Models for Exascale" (under submission), 2013.
- W. Ma and S. Krishnamoorthy, "Data-driven fault tolerance for work stealing computations," in Proceedings of the 26th ACM international conference on Supercomputing, pp. 79–90, ACM, 2012.
- Roger Pearce, Maya Gokhale, Nancy M. Amato, "Scaling Techniques for Massive Scale-Free Graphs in Distributed (External) Memory, IPDPS 2013.