Project Ideas Brainstorming

Ioan Raicu
Computer Science Department
Illinois Institute of Technology

CS 595: Data-Intensive Computing
September 21st, 2011

Developing a research proposal

- Identify a problem
- Review approaches to the problem
- Propose a novel approach to the problem
- Define, design, prototype an implementation to evaluate your approach
  - Could be a real system, simulation and/or theoretical
- Write a technical report
- Present your results
- Write a workshop/conference paper (optional)

Distributed Operating Systems

- Distributed Operating Systems
  - http://www.scalemp.com/
- Achieve a unified OS across machine boundaries
- The opposite of virtualization, which creates multiple virtual OS instances on one machine
- Choose an OS to modify
  - CPU scheduler ➔ load balancing
    - Modify the OS scheduler to be aware of threads and cache locality
  - Memory manager ➔ shared memory
  - File system ➔ leverage shared/parallel file systems
- Choose a virtual machine to modify (e.g. Java)
- Evaluate workloads for performance and scalability

Virtualization Impact for Data-intensive Computing

- Virtualization has overheads
- Quantify these overheads for a variety of workloads
  - Computational intensive
  - Memory intensive
  - Storage intensive
  - Network intensive
  - Across different virtualization technologies
  - Across different hardware
- Survey the latest research in addressing shortcomings of virtualization

Data aware scheduling on erasure codes based distributed file systems

- Distributed file systems use replication to ensure reliability of data
- Replication
  - Pros: Easy to implement, increases data locality and perf
  - Cons: Expensive and inefficient, in terms of network bandwidth and disk space
- Erasure codes:
  - Pros: Efficient in disk space usage
  - Cons: Harder to implement, expensive computationally, decreases locality
- Investigate replacing replication with erasure codes

Distributed Job Management

- Explore decentralization of job managers
- Potential load balancing
  - Load balancing
- Potential solutions:
  - Work stealing
  - Hierachical architecture

Automatic parallelism discovery

- Most code is inherently sequential in nature ➔ this was OK while we doubled processor speeds according to Moore's Law
- Multi-core and manycore architectures are making sequential codes inefficient
- How to parallelize existing codes without burdening the programmer

Manycore Architectures

- 100~1000 cores per GPU
- Does cluster computing programming approaches apply to GPUs?
- How can GPUs be generalized for HPC use?
- Does MapReduce map well to GPUs?
- What architecture support is needed?
  - Cores should have L1/L2 caches, and GPU memory should be a L3 cache for the host memory ➔ Nvidia Fermi might be a step in the right direction
  - Allow cores to execute independent kernels
  - No enforcement of coherency across cores
  - Allow core-to-core communication

GPU and MICA

- Workflows on GPUs
  - Make GPUs look like clusters of computers
  - Enable scheduling of independent tasks with support for file I/O
  - Intel MICA architecture might be useful
- Virtualize GPUs
  - Allow GPUs to be partitioned over multiple virtual machines
  - Work by Peter Dinda might be relevant and useful
- Applications on GPUs
  - Medical imaging, Astronomy

Data-Intensive File Systems

- Implement a distributed file system
  - Use of FUSE for a general POSIX interface
  - Use structured distributed hash tables for distributed meta-data management
    - Can scale logarithmically with system size
    - Can create network topology aware overlays
- Relaxed data access semantic to increase scalability
  - eventual consistency on data modifications
  - write-once read-many data access patterns
- Evaluation scalability and performance
  - Compare to NFS, GPFS, PVFS, Lustre, HDFS

Storage

- Understand file access patterns in HPC
  - How large are files and directories, how often is data accessed, what data is write once read many, or even write once read never, how much data is modified concurrently, etc…
- Explore the use of FUSE to implement various file systems functionality not being met by existing file systems
- Adding Data Provenance support to distributed/parallel file systems
- Adding novel features to filesystem namespace

PVFS

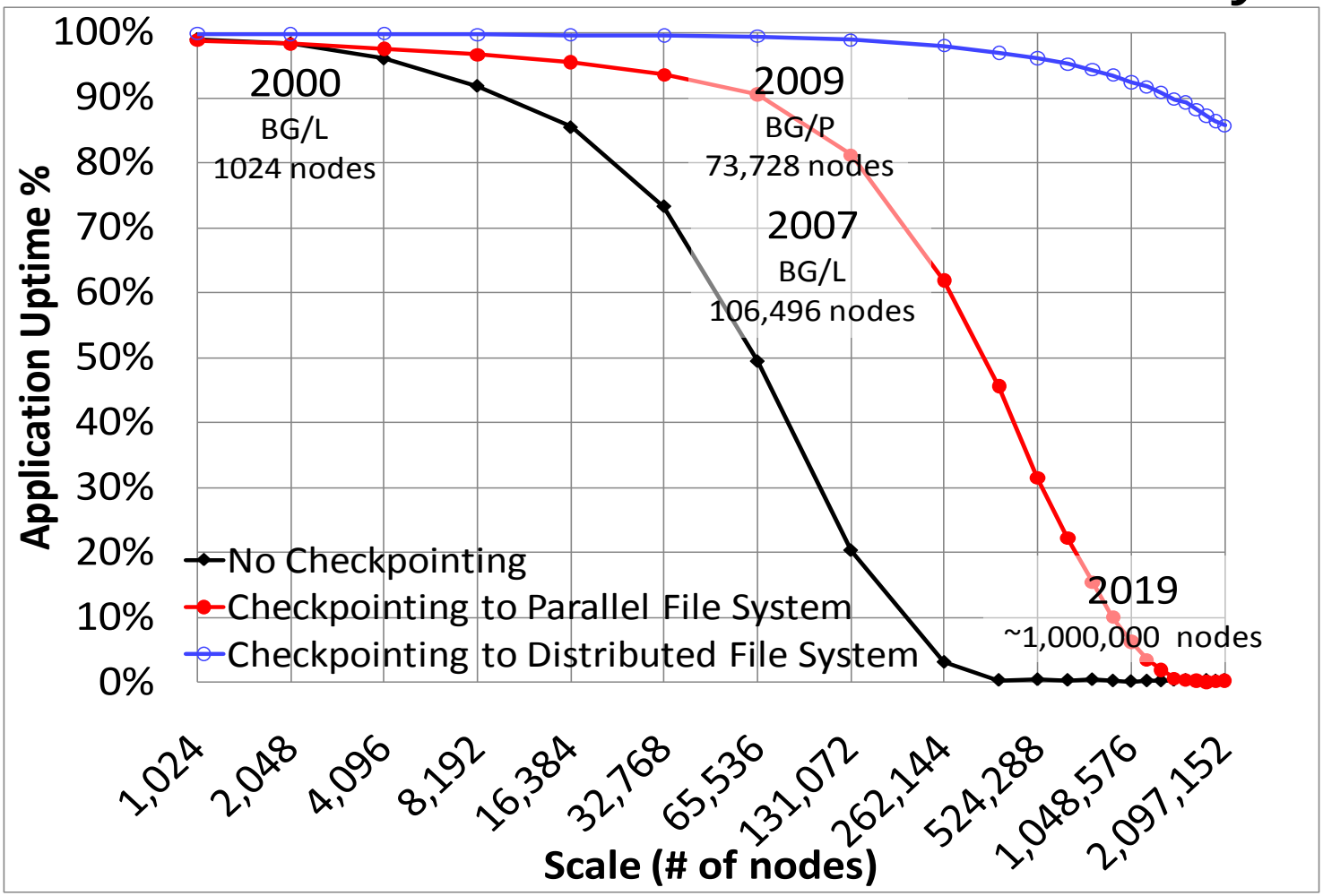
- Modify the open source PVFS to achieve improvements in various areas:
  - Fault tolerance
  - High availability
  - Metadata performance
  - Scalability
- Compare PVFS to GPFS and Lustre for various workloads

Cloud Computing

- Compare cloud performance with grids and clusters
- Explore variable pricing schemes, utilization models, etc
- Reducing the cost of cloud storage through novel architectures

Reliability

- Simulations at exascale and reliability



Reliability

- Reducing Checkpoint Overhead for MPI Applications
- Exploring redundancy to optimize MTTF and system throughput

Virtual Replicas in HPC Systems

- High failure rate in modern HPC systems
  - Large number of components
  - Use of off-the-shelf unreliable components
- Failure rates dynamically varies based on
  - System architecture and Workload
- Replication for fault detection (possible tolerance)
- Independent virtual machines as replicas instead of stand-alone nodes

Benchmarking

- Distributed file systems benchmarking
  - Compare PVFS, GPFS, FusionFS, HDFS, and others
- Distributed hash tables benchmarking
  - Compare Chord, Tapestry, Kademlia, C-MPI, memcache, and some database-centric systems

Questions