Project Ideas Brainstorming

Ioan Raicu
Computer Science Department
Illinois Institute of Technology

CS 595
Hot Topics in Distributed Systems: Data-Intensive Computing
September 13th, 2010

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
- 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
  - 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

- Goal:
  - Maximize data locality in applications data access patterns
- Approach:
  - Move application to data
- Potential problems:
  - Load balancing
- Potential solutions:
  - Move data to application sometimes
- Involves data-aware scheduling algorithms and analysis

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

HPC and Scientific Application on 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

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

Data Staging in Data-Intensive Computing

- Most compute nodes are in 1 or 3 states
  - Input, compute, output
- Blocking I/O can yield low processor utilization
- There is a need for transparent mechanisms to overlap I/O with computations
- Project involves working and possibly modifying with HPC and MTC middleware (e.g. MPI, Swift, Falkon)

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

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

- Explore Cloud Computing to construct turn-key clusters with various software stacks
- Compare cloud performance with grids and clusters
- Explore variable pricing schemes, utilization models, etc

User Level File Systems

- Explore the use of FUSE to implement various file systems functionality not being met by existing file systems

Operating Systems
Cache Aware Scheduling

- Modify the OS scheduler to be aware of threads and cache locality

Questions

?