CS 550:
Advanced Operating Systems

Project Ideas Brainstorming

Ioan Raicu
Computer Science Department
Illinois Institute of Technology

CS 550
Advanced Operating Systems
January 27th, 2011

Developing a project proposal

- Identify a problem
- Review approaches to the problem
- Propose an approach to a solution
- 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

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

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
  - Investigate work stealing algorithms for load balancing in distributed job management

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

GPU Computing

- 100~1000 cores per GPU
- Implement and compare various applications on GPUs and CPUs

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

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

TCP Performance

- TCP performance is sensitive to latency
- Tune TCP to perform better over high latency links
- Implement reliability over UDP to offer better performance
- Compare to UDT
- Find optimal number of TCP streams automatically

Checkpointing

- Checkpointing is used to implement reliability
- Investigate novel approaches to achieve reliable and fast checkpointing

MapReduce

- Implement various applications on MapReduce (Hadoop) and benchmark their performance
- Compare to other MapReduce frameworks (Sector/Sphere)

Sort

- Implement a distributed sort
- Benchmark it on large datasets

Web Server

- Implement a multi-threaded/process web server and compare its performance to Apache

Web Services

- Benchmark the performance of various web service implementations

Monitoring

- Implement a distributed monitoring system
- Compare to existing ones (Monalisa, Ganglia, etc)

Questions