

CMSC 714

Lecture 21

Parallel I/O

Alan Sussman

Notes

- Group Project presentations next Tuesday/Thursday
 - email sent with (approximate) schedule today
 - final report due Wed., Dec. 12
- Course evaluation web site open
 - <https://www.CourseEvalUM.umd.edu>
- Exams returned today
 - Mean: 78 StDev: 10

2

IBM GPFS

- Designed to support high throughput parallel applications, including multimedia
 - well suited for scientific computations
 - still used in many of Top 500 supercomputers
- Main idea is to use parallel I/O to increase performance and scale to large configurations
 - increase bandwidth by spreading reads and writes (even to a single file) across multiple disks, especially for sequential access
 - avoid the “one file per parallel process” model, or sending all I/O through one node
 - use internal high performance switch, plus separate I/O nodes, for I/O from parallel processes running on nodes
 - files can be both striped across multiple I/O nodes, and across multiple disks in each I/O node

3

IBM GPFS

- Each node runs a demon (mmfsd) to provide I/O services
 - one demon runs a *metanode service*, to serve file metadata (ownership, permissions), and inode/directory updates
 - one demon runs a *stripe group manager*, to keep track of available disks
 - a *token manager* to synchronize concurrent access to files, maintain consistency across caches
 - each application node demon mounts a file system and performs file accesses (through switch, to I/O nodes that have the disks with the data)
- Client-side caching
 - inside Virtual Shared Disk (VSD) layer in kernel (server is on I/O nodes)
 - *pagepool* in each application node's memory
 - read-ahead discovers sequential and constant stride access patterns
 - write behind allows application to continue after data copied into pagepool – cost is extra copy to pagepool
- Experiments show that GPFS scales well to very high absolute performance for sequential accesses
 - need big transfer sizes for non-sequential accesses to get decent performance – use MPI-IO to aggregate (collective I/O)
 - 1 server can handle up to maybe 6 clients – this is technology dependent (switch, disks, processors)

4

Active Disks

- Goal is to move the computation to the data, by offloading processing to disk resident processors
- Motivation is that even fast host processor will be unable to keep many disks busy if it's doing any serious processing of the data
 - you say MapReduce, but why not do MapReduce at the disk?
 - in later work, just attach the right number of disks to a host (technology dependent), and do processing in host
- Stream-based programming model
 - *disklets* that read from one or more input streams, write to one or more output streams
 - disklets configured and controlled from host, and have limited capabilities, to protect against errors or malicious code
 - read data into fixed sized buffers (chunk at a time)
 - no dynamic memory allocation
 - I/O ops initiated from host program

5

Active Disks

- Applications include data warehousing, image processing, satellite data processing, ...
 - examples of how to write disklets given for all of those
 - performance comparison against host only programs with conventional disks
 - not completely fair, since the Active Disk implementations use processing power from multiple disk processors
 - but each disk processor is less powerful
 - simulation-based experiments, but fairly detailed, accurate simulations – used multiple datasets, quite large for the time, data striped in large chunks across disks (256KB)
- Experiments show that Active Disks scales well with more disks, performs better than conventional architecture when significant processing on data is required
 - puts much less stress on network between disks and host than conventional architecture
 - host can become a bottleneck when used for collecting and redistributing data from multiple disks

6