Middle-ware, Grids, and Distributed Operating Systems in Scheme to maximize overall computing power

Dan Beatty

February 13, 2005

1 Distributed Operating Systems

One way to look at distributed operating systems is to compare the actions to that of a conventional operating system. One feature is the scheduler and process management. Typically, most distributed systems such as Globus and Avaki approach process management from a point of view of a batch system. While batch systems provide a simple and a convenient scheme that many users are familiar with, its simplicity lacks many features that are necessary in larger operating systems.

For example, take Linux's task structure which has the job of keeping track of the tasks running. This structure contains an process ID table, the doubly linked list task list, task states, processor which the process is running on, counter for the time slice, priority, policy, real time priority, need rescheduling, and task queues. Most other operating systems have these properties as well. What a distributed operating system may need for its basic structure:

- Process ID
- Process Priority
- Task state
- Processor that the process is running on.
- Real time priority
- Need Rescheduling.

The task queues and task lists are simply collections of these structures and are shared amongst the collective distributed operating systems. Counter values are values that kept at the node level, but should be used as a means of reporting on a task in cases of good, bad or indifferent. The action of the node's operating system could be. These reported conditions could be interruptible task, need reschedule, and goodness. As far as the scheduling of the tasks, next, previous or present it is enough the for the distributed operating system to allow the the node to handle this and keep

track of the load for the collectives use. Two additional cases exist as to whether the node is being operated on directly by a user or not.

Some issues for schedulers, especially real time schedulers:

- Swapping in and out of hard disk space: How does this affect processes planted by a grid system, and how does this show up in performances metrics for the collective system?
- Interrupts: If it is possible that a user can gain access to a node in the system and start using the node, how does this affect the performance? How do interrupts get relayed in for remote devices?
- Aperiodic or reactive tasks with deadlines.
- The general non-real time process tasks such analysis or long time computation without a specific time constraint.
- All dynamic planning must incorporate order preserving computation.
- Priority driven scheduling: What consists of a priority task? How are pre-emptions called for in cases where priorities such as real-time, or other deadline tasks arrive.

2 Previous conventions

2.1 Conventional Thinking

In case a task requires preemptive action, then this implies task migration. By definition [**?**], the transfer of these tasks that have not completed is what is called task migration. Task placement is the transfer of tasks that are yet to begin execution. In the case of task migration, a preserved copy of the tasks state is transfered as well. Task migration is justified under the following conditions:

- when the occurance of high system load disrupts service to users.
- When most of the load originates at few nodes.

In these case, receiver-initiated task transfers improve performance. Two schemes that claimed to such a thing in distributed form both had the Mach micro-kernel which were NeXTStep and MOSIX. However, details on how NeXTStep did this still remain vague. Zillion for example does not appear to have such task migration or check-pointing. However, task placement is handled by transferring of bundles representing the task and supporting libraries. Also, it is unclear as to how to get at those pages of memory.

Placement policy is an interesting question, but goes by one simple philosophy. Define a receiver if it is one of the most lightly loaded nodes in the system. Otherwise it is a sender node.

2.2 Conventional Scale of Thought

One typical convention of thought that is applied to distributed computing is how far distributed the systems are spread apart. Some systems may be within the same domain both logically as well as physically. Having said this, a domain can be defined in terms of transfer speeds and latency relative to elements in that domain. Obviously inter-domain communications have larger latencies even if transfer speeds are equivalent.

There are few consequences for this latency issue. If systems are within a small latency domain then issues about clocks become less of a problem, and precision on that clock can be much more precise.

Examine the classical thinking on distributed operating systems. The issues are on event ordering, distributed mutual exclusion (DME), centralized DME, Fully Distributed DME, transaction management, locking protocols, deadlock prevention, deadlock detection, and resource ordering. However, there is one underlying concept to the classic approach and that concept is latency of signal and the cost of synchronization.

Super-computers handle this problem by being relatively close together. Any cluster with the relative transfer speed and small latency of a super computer can be considered to have the same characteristics. As latency increases and/ or relative transfer speed decreases, the ability of the nodes to behave as one diminishes, and features have to simplify to accommodate the mere capabilities of the collective. Thus any network operating system whether it call itself a Grid or middle-ware must take these characteristics into account to maximize computing power.

3 Consequences of Conventional Thought

3.1 Event Ordering

“

- Establish that A preceded B, even if the A and B are on different machines
- Timestamp each system event with a logical counter
- Advance the counter with each event
- Advance the counter when a message with a later time stamp comes in
- Place site ID in low order bits to prevent one site from always having a later TS

” A real time system can vary on campus.

3.2 Distributed Mutual Exclusion

“

- Assumptions
  - The system consists of n processes; each process P_i resides at a different processor.
  - Each process has a critical section that requires mutual exclusion.
- Requirement
  - If P_i is executing in its critical section, then no other process P_j is executing in its critical section.

”

3.2.1 Centralized DME

- Chose a coordinator
  - A single system acts as a coordinator.
- To enter critical section send a message to the coordinator and await a response
- Coordinator responds only when the requesting process can enter
- On exit, send coordinator a completion message
- 3 message per critical section entry

Difficulties with the centralized approach are bottlenecks at the server. On the bright side, deadlock is easy to detect and handle in the this scheme using algorithms like banker's algorithm on the coordinator.

3.2.2 DME: Fully Distributed Approach

“

- To establish critical section, P_i generates a new time stamp (P_kTS) and sends it to all other processes
- Upon receiving a request, P_k responds, or defers its response
- When P_i receives a reply from all other processes, it can enter its critical section.

- After exiting its critical section, the process sends reply messages to all its deferred requests.
- Whether P_k replies immediately (P_kTS) or defers is based on three factors
  1. If P_k is in its critical section, then it defers
  2. If P_k does not want to enter its critical section, then it sends a reply immediately to P_i
  3. If P_k wants to enter its critical section but has not yet entered it, then compares its own request time-stamp with the time-stamp TS.
     - If its own request time-stamp is greater than TS, then it sends a reply immediately to P_i (since P_i asked first).
     - Otherwise, the reply is deferred.

”

3.2.3 Fully Distributed DME Good Points

“

- Freedom from deadlock is ensured
- Freedom from starvation, since the time-stamps ensure a FIFO schedule.
- The number of messages per critical section entry is $2 \times (n - 1)$ minimum fully distributed processes acting independently

”

3.2.4 DME Bad Points

- Processes need to know each other, makes adding new processes difficult and costly
- If one process fails it can stall the rest
- Processes not entering their critical section take a performance hit as they respond to requests.
- Best suited for small, stable sets of processes (the easy case)

”

3.3 Transaction Management:

Transaction management deals with the need to have all sites commit some action, or have none of them do so. Each site runs a transaction manager which manages its role in such a transaction. One manager acts as the transaction coordinator for the given transaction. “

- All parts of the transaction are performed and committed or none are
- A central transaction coordinator handles:
  - Starting the transaction (and recording the fact)
  - Assigning sub-tasks to appropriate sites, which my not be homogenous
  - Coordinator successful completion or notifying participating sites of failure
- Each site has its own transaction manager, and may act as a coordinator

”

3.3.1 Distributed Transaction Algorithm

“

- The coordinator, C_j, adds <prepare T >record to its log, and sends the messages to all sites
- When a site receives <prepare T>, its own transaction manager determines if it can commit the transaction
  - No add <no T> to log and respond <abort T>
  - yes
    - add <ready T> to the log
    - force all log records for T onto stable storage.
    - transaction manager sends <ready T> message to C_i
- When all ready messages are received, C_j writes <commit T> to its log.
- Coordinator records decision on stable storage, and sends the decision to all sites.
- Coordinator collects response, and decides to commit if the response is ready, otherwise decision is to abort
- Sites take whatever action that they need to.

”

3.3.2 Failure Handling

“

- Log has <commit T> record, redo (T)
- Log has <abort T> record, undo (T)
- Log has <ready T> record ; consult coordinator for result
- If the coordinator is down, query other sites
  - Any site with a <commit T> or <abort T> record indicates the coordinator made that decision
  - If some sites do not have <ready T>, coordinator must have decided not to commit, so abort
  - If all sites have <ready T>, we must wait for coordinator recovery.

”

Size of the directory ? Assign what to the inode? File size 12 to 1024? Look up for knoppix. Depends on the file system type. Can then be arbitary as long as it works.

4 Locking in a distributed system:

Various data resources (such as rows in a database) may need to be locked in a distributed system. This problem is similar to the critical section problem, but needs to account for the possibility of replication and deadlock.

4.1 Locking Protocol:

- Sites have data which they need to make available to others, but need to allow only one writer at a time
- Similar to critical sections, except
  - Deadlock detection and prevention
  - Possibility of replicated data

4.2 Simple Locking Protocols:

“

- Single Coordinator: One coordinator controls access to data, regardless of where the data lives on the network. Messages sent are minimal in most cases, but lock and unlock messages must be sent even if the resource is local.
  - Potential bottleneck and single point of failure
  - Deadlock detection and prevention easy with the banker's algorithm
  - Multiple read locks are allowed, but only one write lock is allowed.
- Multiple Coordinator: Each site manages its own data
  - Eliminates the bottleneck
  - Deadlock is difficult to prevent, and must include a distributable scheme.
- Majority Approach: Lock requests are sent to all sites. The lock is considered to be held when a majority respond saying they are not locking the data.

"

Naming Schemes:

4.2.1 Majority Protocol:

"

- Data is replicated at various sites
- Locking process must receive lock from a majority of sites replicating the data
- Minimum $2(\frac{n}{2}+1)$ messages for lock request, $(\frac{n}{2}+1)$ message for unlock
- Dead lock prevention even trickier
  - Consider a system with $2n$ site: deadlock is two different lock requests each get n responses

"

4.2.2 Bias Protocol

- Data may be replicated at many sites
- A shared (or read-only) lock can be obtained from one site
- An exclusive (or write) lock requires the lock be obtained from all sites replicating the data

”

Caching Scheme is the primary thing for many of these DFS

Deadlock schemes in the distributed system:

Resource acquistion is done by priority and

4.3 Deadlock prevention

There are three general schemes for preventing deadlock in a distributed system. We assume each process has a unique time stamp, and that this time stamp remains the same even if the process must be restarted (rolled back).

“In this case a, roll back means the process must return to a point where it is not using the resource. In practical terms, this almost always means terminating the process and restarting it; although certain systems, mostly in cluster or mainframes, allow for checkpointing. Checkpointing basically saves a programs state periodically, and lets a program revert to that state. ”

“

- Assign a global priority to each resource, force resource to be requested in order
  - Requesting a resource with a lower priority than the highest one currently held requires releasing and reacquiring
  - This is the same as the non-distributed scheme
- Central coordinator could clear all requests, run the banker’s algorithm

”

4.3.1 Resource Ordering

This is the same as as the single processor deadlock prevention algorithm. All resources are ordered, and processes must request resources in order. If they need a resource with a number lower than their highest numbered resource, they must first release the higher numbered resource. This prevents the circular wait condition required for deadlock.

4.3.2 Wait - die scheme

“

- Each process P_i gets unique priority number
- If P_i requests a resource held by P_j
  - P_i waits if it is older
  - P_i dies if it is younger
- Processes keep their original time stamp
- Older processes tend to wait
- Younger processes may die several times before they get a resource
- Minimum deadlock and message passing

”

4.3.3 Wound - wait

“

- Pre-emptive technique
- If P_i requests a resource held by P_j
  - If P_i is younger than P_j, then it waits and P_j is rolled-back
  - If P_i is older than P_j, then P_j waits and P_i is rolled-back
- Fewer roll backs, because a rolled-back process will always wait
- Roll back is fatal in most systems

”

Killing processes and hoping they will run eventually.

4.4 Deadlock Detection

“

- For simplicity, assume only one instance of each resource, multiple processes per site
- Each site has a local wait for graph
  - Cycle on this graph indicate deadlock

  - Lack of cycles do not indicate that the system is not deadlocked
- Cycles on the unknown global wait for graph indicate deadlock

" Example: Local graphs

4.4.1 Central Deadlock Detection

- Local sites periodically communicate their wait for graphs to the coordinator, which checks for deadlock
- False cycles may occur if there is a lag between processing of releases and acquisition requests

4.4.2 Distributed Deadlock Detection

“

- Each site maintains a local graph, with a node P_{ex} (external)
  - Edges to and from indicates resources held by or request from other sites
- Cycles involving P_{ex} may or may not indicate deadlock
- If a cycle invovles P_{ex}, that cycle is sent to the appropriate site, which updates its graph and repeats.

4.5 Election Algorithm:

“

- When a coordinator fails, the system needs to choose a new one
- Failure can be detected in various ways, usually through repeated time-outs being reached on requests
- Two approaches: bully algorithm and the ring algorithm

”

4.5.1 Bully Algorithm:

“

- When P_i find the coordinator is down, it attempts to elect itself the new coordinator
- P_i sends an election message to every process with a higher priority, then waits
- If no response is received, P_j notifies all lower priority process that it is the new coordinator
- If an election message is received from a lower priority process, the receiver notifies the sender, and then tries to elect itself using the same algorithm.

”

4.5.2 Example: Bully Algorithm

“

- P_3 detection failure of P_0 attempts to elect itself
  - Sends election message to P_1 and P_2
- P_1 and P_2 veto election of P_3, and attempt to elect themselves
  - P_2 sends election message to P_0 and P_1
  - P_1 vetoes the election of P_2
  - P_0 fails to respond, and P_1 wins the election
- P_1 notifies all lower processes that it was won

”

4.6 Ring Algorithm

- Sites are arranged in the logical ring, and all messages (at least for the election) pass in only one direction (to the right)
- Each site maintain a list of all other active sites, the ring election algorithm rebuilds the list
- If process P_i detects a coordinator failure, it creates an empty active list, and sends its neighbor an elect(i) message
- When P_k receives elect(i) from the left, it must respond one of three ways:

  - If this is the first elect message seen or sent, create a new active list with i and k. Sends elect(k), then elect(i).
  - If $i \neq k$ add to active list, forward elect(i)
  - If $i = k$, then we have received our original message back, active list now complete.
- With a complete active list we can choose a new coordinator

5 Process Management (Darwin)

One possibility: Service Oriented Programming could be used as a means to optimize the process management scheme. What would be needed?

- A Service Library
- A comparable computational model to any other Algorithm on a Turing Machine.
- A comparison to procedural models:
  - Data
  - Method
  - Control
- Or Object Oriented Models
  - Object
  - Control

However, this does not answer the Application and Object Paradox.

- How do we manage applications for zero configuration and zero install?
- How do we manage remote resources?
- How do maximize compute power?

Part of the Application and Object Paradox is the nature of many batch submitted jobs such as numerical simulations. One example is in MPI programming. The basic idea is set an army of program drones that talk with each other to accomplish a task. However, the what is the application. In many respects, the MPI batch engine could be seen as the actual application since it sets the work in motion. Now consider the average application. Every bit of work is set in motion by the user operating on the application and the application calls any method necessary to fulfill the users wish.

If the desire is to maximize computational power, then a model needs to add a more techni-color spectrum to the idea of computational spectrum. Up until now, the idea computing hardware was

either server or desktop. This black and white picture has been portrayed with commodity machine and super-computer and very little gray area in the middle. The consequences to maximized computation with data includes:

- Discovery Services
- Look Up services
- Secure access to the data
- Confidence that proper services are being called.
- Accessible common file system
- Platform issues:
  - Platform Dependent Code
  - Non-platform dependent code such as JVM, .net, and intepreted languages.

Questions should be, are there any API's needed? How does one use distributed objects? How does one present this concept in a way that is not threatening to the average consumer?

5.1 Darwin and Distributed Objects

OSX is a product of NeXTStep with the Mach micro-kernel. As such it also has NSPorts. One feature that is also present is another service registration scheme called Rendezvous. Rendezvous is Apple's implementation of Zeroconf DNS which allows services to declare the name, type, port, etc.

OSX uses NSPorts to provide distributed objects (DO(s)) and uses the run loop and or thread to achieve a non-blocking solution. Such DO are called via normal message passing routines associated with Objective C and NS Objects. This mechanism provides a sort of proxy for which there are two classes" NSDistributedObject and NSProxy.

An NSConnection object has two instances of NSPort: one receives data and the other sends data. An NSPort is a superclass to all other ports. NSMachPort uses Mach messaging and is typically used solely on the machine itself. NSSocketPorts use socket to go between machines.

There are addition identifier/ modifier types applied to distributed objects: functions, methods and members alike. These key words are as follows:

- oneway void (client does not wait for a response.
- in (A receiver is going to read the value but not change it.)
- out (A value is changed by the receiver by not read)

- inout (receiver is to both read and write the value).
- bycopy (argument is archived before sent and de-archived in the receiver's process space)
- byref (the argument is represented by proxy).

Each connection can have a delegate. Each time the connection spawns a new "child" connection, the "child" will have its delegate outlet set to point to its parent delegate. The connection monitor is a class for logging delegates and their connections.

5.2 Distributed Tasks

Of course, there is nothing wrong with calling distributed tasks either. An example was provided by O'Reilly's articles and written by Drew McCormack May 11, 2004 [**?**]. This analysis examines the crucial parts.

Apply Filters is the method that calls Distributed Task. There are many nuggets of value in addition to the calls for:

- Add Sub Task with Identifier . This call includes
    1. The identifier
    2. Launch path
    3. Working Directory
    4. Output Directory
    5. Standard Input
    6. Standard Output
- Launch

The methods of how "Photo Industry" provides these values are somewhat interesting.

- The first section of Apply Filters acquires the time.
- Next initiates local instances of the file manager.
- The output directory is fed into Apply Filters and is not interesting.
- The temporary directory segment is interesting.
    1. It uses the processes own information (supplied by NS (OSX) which identifies the process in all of its details. The way this is used to access programs is with in the application itself.
    2. The temporary directory of functions which acquires the temporary directory as specified by the OS. (Any where NS applies).
- The next section claims to produce standard input for the filters which are actually programs and the parameters to those programs. The means for this is the typical array/ dictionary scheme of Objective-C.
- The next segment produces input and temporary directories for the input data (the photos). Features of these production(s) is the production of directories for the sub-tasks. Thus a scheme for dividing the work judiciously is being applied.

The question of the thread oriented submission becomes an issue.

Also, the feeding of data structures becomes an issue for the parent application:

- The manner the sub-tasks are divided up as a list of files (input). Items copied into these directories into these directories are the data (photos) and the programs to work on them.
- These structures include message forwarding which is the purpose of a NeXTStep delegate.
- "A delegate is an object directed to carry out an action by another object." page 456 [**?**]
- Once the sub-tasks and its data are determined, the sub job is copied out of the bundle (app), and the executable (script), then the sub-task queue is loaded.
- The rest of the methods are delegate methods.

5.3 Small Question

- Can xGrid be adapted for distributed process submission and management?
- Can xGrid provide means to identify the owner of such a job?
- How does xGrid handle processes and threads?
- Can xGrid advertise or show its process tables and subsequent to other Grid middle ware?
- Can xGrid advertise or show which processes can run and where?
- How are "safe nodes" handled?
- How are remote objects published in temporary fashions?
- How to distribute threads and parallel tasks?
- How to control jobs at a user level?
- How to deploy apps such that when a user starts them, and return to those application after leaving and coming back to it on a different terminal? Note the idea is that the application should still be working while in this nebulous space.

5.4 Big Questions

What contributions on this xGrid idea are worthy of a Ph.D. dissertation?

What synergy can be tapped to make a more complete product?

How can this product be made such that it is non-threatening by design to accomplish this computing maximizing in a non-threatening manner?

6 Calendar of Events

The long range plan includes the following objectives:

1. Plan Courses
2. Set up doctoral advisory committee and title
3. File Program for Ph.D. courses and preliminary exam
4. Courses in any order
    - GIS (done)
    - Intelligent Systems
    - MPI
    - Databases
    - Intelligent Search
    - Compilers
    - Multiprocessors
    - Communication Networks
    - Virtual Reality
    - Fault Tolerant Systems
    - Neural Networks
    - Reinforcement Learning
    - Parallel Processing
5. Qualifiers: Note the qualifiers should occur after the first 24 hours of course work, and the "residency requirement" should be declared done by that time.

One other note, research hours should be used if necessary to arrive at thesis proposal by the time of the qualifiers.

6.1 Proposed research for MPI course

In essence, this project is to expand on Apple's xGrid's capacity of process management to be more comparable to that of a conceptual distributed operating system. Part of this involves comparing MPI message passing to that of Distributed Objects, Rendezvous, and Distributed Tasks.

6.2 Examples: Wavelet Matrix Multiply

6.3 Example: Wavelet for on FITS Images

7 Rendezvous

8 Directory Services

What information does a distributed operating system require? Here are just a few proposed by Core Unix and OSX programming:

A database to contain

- Host configuration
  - IP addresses
  - Shared directories
- User information
  - User certification (identification)
  - Contact information
  - Broker's Contact information.
- Brokered service should be an open standard.
- Should be distributed with clearly defined entities.
- Insurance from prying.
- Insurance from tampering
- Good Performance and reliability.

A few points in reference to Core Unix and OSX programming are as follows. Encryption to ensure confidentiality is a good idea. Also, the use of public keys to prevent tampering has its merits especially in a system that can not ensure that all keys are kept secret at all times like military communications security. Because this topic represents a facet for control, it becomes political between competing proponents. Apple produced a framework called Directory Services which is include with Netinfo.

8.1 Concepts

Such a directory is typically a hierarchical database containing entities called records with attributes. Typically, a hierarchical database identifies its entities by both location in the hierarchy and unique feature in that hierarchical grouping. Examples of this structure are MS Windows' registry and OSX's Netinfo directory. One feature provided by Apple's Directory Services to developer are pseudo-nodes, and many of these can be used. Pseudo-nodes are allowed to query for authentication information, users, groups, and aliases. Writing is a privileged activity which requires authentication as that privileged entity.

Data structures with Open Directory and Directory Services are as follows:

- type names start with the prefix "t"
- field names start with the prefix "f"
- function names start with the prefix "ds"
- List like structures are indexed starting at one.

Data Structure expansion

- tDirReference
  - Keeps track track of the information for the active session
  - First argument for most Open Directory API functions.
  - Created by dsOpenDirService().
  - Closed by dsCloseDirService().
  - During the session many different nodes can be accessed on many different directory servers.
- tDataBuffer
  - Is primarily used as a string
  - Is actually a structure defined in terms of buffer size, length, and buffer data
  - Allocated by dsDataBufferAllocate
  - Released by dsDataBufferDeAllocate
- tDataList
  - A list of data buffers
  - For component based data buffers
  - It may be released by dsDataDeallocate
  - dsBuildFromPath is an example function that may allocate such a data list
- tDirNodeReference

  - Provided when a node is opened
  - Identifies uniquely each node

- tRecordEntry and tAttributeList
  - Each record provided by any node in the Directory Service contains a list of attributes
  - Searches for records are based on type of record, and which record is desired.

- tAttributeValueList, tAttributeEntry, and tAttributeValueEntry
  - A record may have many attributes
  - An attribute may have a name and many values.