Lecture 6: Locus

Locus

1. QUESTIONS FROM REVIEWS:
    a. No motivation for transparency
2. Overview
    a. QUESTION: What is the goal of Locus?
        i. Build a distributed system that acts like a larger monolithic system
        ii. Build a system that runs unmodified Linux programs in a distributed environment
        iii. Environment:
            1. Cluster of minicomputers
            2. Supports many more users than machines
            3. Any single machine could not handle workload
        iv. Make distribution **transparent**
            1. Programs cannot tell when remote resources are used
            2. Single global name space
            3. Process control (signals, IPC) works globally
            4. Failures of remote nodes handled automatically, invisibly to programs
            5. QUESTION: What about performance transparency?
        v. QUESTION: How close do they get to transparent?
        vi. QUESTION: is transparency a good goal?
            1. There are performance differences, so a program cares about distribution. E.g., latencies for timeouts in failure, added latencies for communication
3. QUESTION: What are the design goals for Locus
    a. Keep Unix syscall interface
        i. Provide per-process variables that are inherited to control policies
            1. Destination for replicated files, number of copies
            2. Destination for fork/exec
    b. Provide **strong consistency**
        i. Read of data should always return most recently written data
    c. Provide **availability**
        i. If data is available somewhere reachable, should be able to access a file
    d. Provide fault and **partition tolerance**
        i. Detect network partitioning, allow local actions with partition
        ii. Keep running in face of node failures
4. Big picture: a distributed OS
    a. Really just one OS instance, but runs on multiple computers, like running Linux or Windows on multiple cores
    b. Close coupling within a computer, loose coupling between computers
        i. Routing for communication
        ii. Failure handling

5. Transparency
   a. Data transparency
      i. Allow transparent access to remote data
      ii. Benefit: allows use of remote data resources
      iii. NFS is (largely) data transparent
      iv. NOTE: users may know from namespace what is remote
   b. Process access transparency
      i. Local resources accessed with same mechanisms as remote resources
      ii. Benefit: user doesn't need to worry what's local and what's not
      iii. NFS, RPC are process access transparent
      iv. WWW is not process access transparent
      v. NOTE: tends to ignore performance differences
   c. Location transparency
      i. Where resources are located is invisible
      ii. Benefit: resources can be moved without disruption
      iii. RPC can be location transparent
      iv. WWW is not location transparent
   d. Name transparency
      i. A given name has the same meaning throughout the distributed system
      ii. Benefit: same name gets to same resource from anywhere
      iii. Fully qualified WWW names are name transparent
      iv. /tmp in most distributed FSes is not
   e. Control transparency
      i. Control of system resources is transparent to its users (e.g., remote processes controlled like local)
      ii. Benefit: easier control of distributed applications
      iii. Locus provides control transparency on processes
      iv. Typical UNIX network of workstation does not provide it on processes
   f. Execution Transparency
      i. Allows processes to execute on any machine in system (and more, perhaps)
      ii. Benefit: easier handling of distributed applications, load balancing
      iii. Java is execution transparent (not load balancing, though)
      iv. NFS provides no execution transparency
   g. Performance transparency
      i. Users don't notice difference when something must be done remotely
      ii. Benefit: if achievable, frees user of worrying about costs of going remote
      iii. NFS has high degree of performance transparency
      iv. WWW often does not
      v. NOTE: How can you do this?
         1. Make local case slow
         2. High speed networks, very powerful servers (e.g. ATM, Myrinet)
   h. Benefits of transparency
      i. Easier software development

            1. System handles all details of distribution
        ii. Support for incremental changes
            1. Changes at network level invisible to apps,
        iii. Potentially better reliability
            1. System provides it for all apps
        iv. Simpler user model
            1. One way to do everything
        v. Flexibility in resource location
            1. Can do it anywhere
        vi. Support for scaling
            1. Add more servers, replicate more, partition differently
    i. When provide transparency?
        i. In applications (especially databases)
            1. Database may be parallel/distributed
        ii. In programming languages
            1. E.g. RPC for remote access
        iii. In operating system itself
            1. Network file systems, Locus
    j. When can you not provide it?
        i. When it's too complex to provide
            1. E.g., heterogeneous systems
        ii. When you want particular resources
            1. E.g., /tmp
        iii. when remote performance is terrible
            1. E.g., over very slow links
        iv. When there is a security difference
            1. E.g. administrative domains
        v. Must be able to bypass transparency
6. Locus assumptions
    a. High speed network
    b. Various speed processors
7. Locus design elements
    a. Global file name space
        i. Same file names used everywhere, as on a monolithic system
    b. File groups (like a volume) – how used?
        i. Unit of mounting to form the file system name space
        ii. Unit of ordering: each file group has a single site for ordering updates
        iii. QUESTION: Why?
            1. Ensures agreement on where to do updates, what is the latest copy
        iv. Note: have to fully replicate mount points of file groups
            1. Limits scalability, assume changes rarely
        v. File descriptors/Inode numbers unique to a file group, so can allocate independently

c. Replicated files
    i. Each file is replicated at a number of locations (different for each file)
    ii. QUESTION: Why have per-file replication policy?
    iii. Partition file group inode space so can allocate unique inode numbers at each place a file could be created.
d. Transactions for multi-file updates
    i. As files are on multiple computers, can update all or none
    ii. NOTE: need to store old+new data (old in memory)
        1. Limits size of transaction to buffer space available
    iii. QUESTION: Was this a good design point? Violates transparency (not part of Unix), so would it be used?
        1. How would you use it?
e. Version vectors for detecting consistencies
    i. Can determine DAG of updates, detect if one logically is after another or concurrent
    ii. Will cover later in more detail

8. Scalability
    a. QUESTION: What does Locus do for scalability
        i. File replication
            1. Can do reads at multiple locations
            2. Particularly useful for directories high in namespace
        ii. Multiple file groups
            1. Partitioning
        iii. Remote execute for load balancing
    b. QUESTION: What does Locus **not do** for scalability
        i. Protocols involving all nodes simultaneously: partition/merge
        ii. Transparency for remote file descriptors
        iii. Per-file replication state
    c. QUESTION: Compare to Condor, 5-7 years later
        i. How differ in goals?
            1. One assume heterogeneity, distributed control
        ii. How differ in approaches
            1. One is loosely coupled, limited transparency
    d. Not have dedicated servers (e.g. file servers, authentication servers)
        i. Could compromise reliability – single points of failure
        ii. May need to cross boundaries anyway for some operations, and having them on different machines adds latency
            1. E.g. authenticating access to a file server
        iii. Locus takes an integrated model: each machine runs all the components of Locus, and do anything (opposite of LARD, Google)

9. Concurrency control / consistency
    a. CSS – current synchronization site
        i. Enforces single writer/multiple reader policy for files
            1. Directories relax this – unlocked reads, atomic inserts/deletes

        ii. Provides locking for files being written
        iii. Knows which files open/closed
    b. Can have multiple CSS if partitioned
        i. Leads to conflicts
10. Replication
    a. Each file/directory has a list of nodes storing data
        i. QUESTION: What if a node joins or leaves permanently?
    b. Updating a file:
        i. Send update to one replica (CSS prevents simultaneous updates at two nodes)
        ii. Replica notifies other copies of change (new version), may push data or not
        iii. Replica (SS) pulls changes using read protocol
            1. On failure, can pull changes later
    c. What happens in the face of conflicts?
        i. If have file locally, can always update and resolve conflicts later
        ii. Favor availability over strong consistency
11. Handling Conflicts
    a. QUESTION: When can conflicts occur?
        i. With partitioning or without?
    b. QUESTION: What constitutes a conflict?
        i. Simultaneous update of a file in two partitions
        ii. Simultaneous operation on **same file name** in two partitions
    c. QUESTION: How handle file conflicts?
        i. Email user
        ii. QUESTION: What do you want here?
            1. Ask program to resolve conflict on open
            2. Provide a program to resolve conflicts when detected
            3. Save both copies under different names
    d. QUESTION: how handle directory conflicts?
        i. QUESTION: What are basic directory operations:
            1. Add a name
            2. Delete a name
            3. Change name/metadata association
        ii. Option A: treat any change to directory as conflicting with any other change
            1. Too many conflicts, restricts concurrency
        iii. Option B: handle each separately
            1. Two files created/renamed to same name
                a. Resolution: Rename both, notify owners
            2. Delete/update
                a. Keep delete unless update after the delete
12. Handling Partitions / Reconfiguration
    a. QUESTION: What is the challenge

        i. Not have full connectivity between nodes
        ii. Protocols assume full connectivity
        iii. May have partitions
    b. Approach: separate problem into two cases
        i. Partition protocol: detect fully connected
            1. Ask your neighbors who they can talk to, AND it all together
        ii. Merge protocol: combine fully connected components that are reachable into a partition
            1. Done centrally at one site
        iii. QUESTION: Why separate the protocols?
            1. Simpler synchronization – have to worry about changes while running protocols
            2. Different needs of the protocols
    c. Major challenges (alluded to but not discussed)
        i. Failures during reconfiguration
        ii. Delays during reconfiguration
13. Failure handling
    a. QUESTION: What failures are handled
        i. Largely a node not being reachable
    b. QUESTION: Who is responsible for handling failed nodes?
        i. Server must handle failed clients by remove state, discarding updates
        ii. Clients must handle failed servers by closing files or trying to open another replica
14. Hard parts about transparency
    a. Fork/exec: bad interface for remote exec, as have to copy lots of address space data
    b. Signals – overall confusing even locally
    c. Shared file descriptors
        i. Requires token-based ownership protocol
15. QUESTION: Where does transparency break down?
    a. Failures not transparent
    b. Conflicts not transparent
    c. Performance not transparent
    d. QUESTION: Is transparency needed?
        i. Compare Condor and Locus
16. Big lessons of Locus
    a. Unix system call interface not designed for transparency
        i. Too much implicit sharing -- file descriptors, signals, pipes
    b. Completely transparent distributed system may not be worth it
    c. Follow on work: V from Stanford, Amoeba from Netherlands, Sprite from Berkeley all drop strong transparency goal
    d. May be more applicable on Multicore, where reliability less of a goal
        i. Barrelfish multikernel
    e. Ultimate use today is to make distributed just the pieces that matter

i. Network file systems on dedicated servers, minimal replication
ii. Remote login
iii. Batch scheduling for CPU/data intensive jobs