Distributed Systems – Theory
11. Distributed Computing Models. Volunteer Computing

Basic models

- The processors in a distributed computing system can be organized in several ways.
- We will look at principal ones:
    1. the workstation model and
    2. the processor pool model,
    3. a hybrid form encompassing features of each one.
- Context: current “on-demand computing”

Workstation Model

- The system consists of workstations (high-end personal computers) scattered throughout a building or campus and connected by a high-speed LAN.
- Some of the workstations may be in offices, and thus implicitly dedicated to a single user, whereas others may be in public areas and have several different users during the course of a day
- In both cases, at any instant of time, a workstation either has a single user logged into it, and thus has an "owner" (temporary), or it is idle.
- The advantages of the workstation model are manifold and clear.
  - The model is certainly easy to understand.
  - Users have a fixed amount of dedicated computing power, and thus guaranteed response time.
  - Sophisticated graphics programs can be very fast, since they can have direct access to the screen.
  - Each user has a large degree of autonomy and can allocate his workstation's resources as he sees fit.
  - Local disks add to this independence, and make it possible to continue working to a lesser or greater degree even in the face of file server crashes.

Workstation model - problems

- The model also has two problems.
  1. As processor chips continue to get cheaper, it will soon become economically feasible to give each user 100 CPUs.
  2. Much of the time users are not using their workstations, which are idle, while other users may need extra computing capacity and cannot get it.
     - From a system-wide perspective, allocating resources in such a way that some users have resources they do not need while other users need these resources badly is inefficient.
- The first problem can be addressed by making each workstation a personal multiprocessor.
  - This is an inefficient use of resources, but as get cheaper & cheaper as the technology improves, wasting them will become less of a sin.
- The second problem, idle workstations, has been the subject of considerable research, primarily because many universities have a substantial number of personal workstations, some of which are idle.
  - Measurements show that even at peak periods in the middle of the day, often as many as 30 % of the workstations are idle at any given moment. In the evening, even more are idle.

Using Idle Workstations

- The earliest attempt to allow idle workstations to be utilized was the program that comes with Berkeley Unix: *rsh*
  - the first argument names a machine and the second names a command to run on it.
  - run the specified command on the specified machine.
  - Although widely used, this program has several serious flaws:
  1. The user must tell which machine to use, putting the full burden of keeping track of idle machines on the user.
  2. The program executes in the environment of the remote machine, which is usually different from the local environment.
  3. If someone should log into an idle machine on which a remote process is running, the process continues to run and the newly logged-in user either has to accept the lower performance or find another machine.

Research on idle workstation

The key issues are:

1. How is an idle workstation found?
2. How can a remote process be run transparently?
3. What happens if the machine's owner comes back?

How is an idle workstation found?

- Idle workstation?
  - At first glance, it might appear that a workstation with no one logged in at the console is an idle workstation,
  - But in many systems, even with no one logged in there may be dozens of processes running, such as clock daemons, mail daemons, news daemons, and all manner of other daemons.
- On the other hand, a user who logs in when arriving at his desk in the morning, but otherwise does not touch the computer for hours, hardly puts any additional load on it.
- Different systems make different decisions as to what "idle" means
- Typically, if no one has touched the keyboard or mouse for several minutes and no user-initiated processes are running, the workstation can be said to be idle.
  - Consequently, there may be substantial differences in load between one idle workstation and another, due, for example, to the volume of mail coming into the first one but not the second.

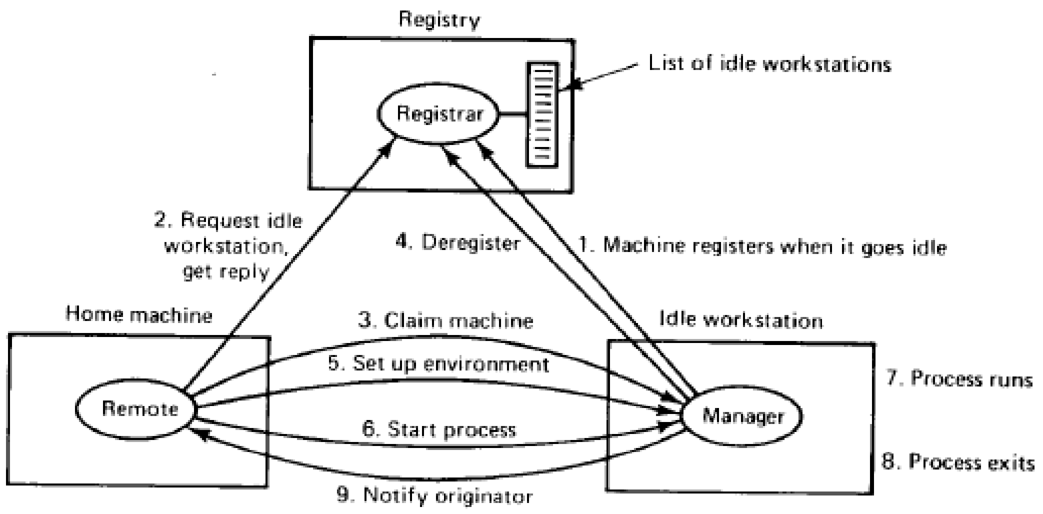
Locate idle workstations

- The algorithms used to locate idle workstations can be divided into two categories:

1. server driven
2. client driven

- Server driven
  - when a workstation goes idle, and thus becomes a potential compute server, it announces its availability.
  - it can do this by entering its name, network address, and properties in a registry file (or data base), for example.
  - when a user wants to execute a command on an idle workstation, he types something like remote command and the remote program looks in the registry to find a suitable idle workstation.
  - for reliability reasons, it is also possible to have multiple copies of the registry.

Server driven



- An alternative way for the newly idle workstation to announce the fact that it has become unemployed is to put a broadcast message onto the network.
  - All other workstations then record this fact.
  - In effect, each machine maintains its own private copy of the registry.
  - The advantage: less overhead in finding an idle workstation and greater redundancy.
  - The disadvantage is requiring all machines to do the work of maintaining the registry.
- Whether there is one registry or many, there is a potential danger of race conditions occurring.
  - If two users invoke the remote command simultaneously, and both of them discover that the same machine is idle, they may both try to start up processes there at the same time.
  - To detect and avoid this situation, the remote program can check with the idle workstation, which, if still free, removes itself from the registry and gives the go-ahead sign.
  - The caller can send over its environment and start the remote process, as shown in Fig.

Client driven

- When remote is invoked, it broadcasts a request saying what program it wants to run, how much memory it needs, whether or not floating point is needed, and so on.
- These details are not needed if all the workstations are identical, but if the system is heterogeneous and not every program can run on every workstation, they are essential.
- When the replies come back, remote picks one & sets it up
- One nice twist is to have "idle" workstations delay their responses slightly, with the delay being proportional to the current load
  - In this way, the reply from the least heavily loaded machine will come back first and be selected.

Key issue 2: Running remotely

- Moving the code is easy.
- To run the trick is to set up the remote process so that it sees the same environment it would have locally, on the home workstation,
  - thus carries out the same computation it would have locally.
  - it needs the same view of the file system, the same working directory, and the same environment variables (shell variables), if any.
- The trouble starts when the first system call, say a READ, is executed.
  - What should the kernel do?
  - The answer depends very much on the system architecture.
    - If all the files are located on file servers, the kernel can just send the request to the appropriate file server, the same way the home machine would have done had the process been running there.
    - If the system has local disks, each with a complete file system, the request has to be forwarded back to the home machine for execution.

Remotely system calls

- Some system calls must be forwarded back to the home machine
  - For example, reads from the keyboard and writes to the screen can never be carried out on the remote machine.
- Other system calls must be done remotely under all conditions.
  - For example, all system calls that query the state of the machine have to be done on the machine on which the process is actually running.
    - These include asking for the machine's name and network address, asking how much free memory it has, and so on.
- System calls involving time are a problem because the clocks on different machines may not be synchronized.
  - Forwarding all time-related calls back to the home machine, however, introduces delay, which also causes problems with time.
- Certain special cases of calls which normally might have to be forwarded back, such as creating and writing to a temporary file, can be done much more efficiently on the remote machine.

=> Making programs run on remote machines as though they were running on their home machines is possible, but it is a complex and tricky business.

Key issue 3: what to do if the machine's owner comes back?

1. The easiest thing is to do nothing, but this tends to defeat the idea of "personal" workstations.
   - If other people can run programs on your workstation at the same time that you are trying to use it, there goes your guaranteed response.
2. Another possibility is to kill off the intruding process.
   - The simplest way is to do this abruptly and without warning.
     - The disadvantage of this strategy is that all work will be lost and the file system may be left in a chaotic state.
   - A better way is to give the process fair warning, by sending it a signal to allow it to detect impending doom, and shut down gracefully (write edit buffers to the disk, close files, and so on).
     - If it has not exited within a few seconds, it is then terminated.
     - Of course, the program must be written to expect and handle this signal, something most existing programs definitely are not.

Migration

3. A completely different approach is to migrate the process to another machine, either back to the home machine or to yet another idle workstation.
   - Hard part is not moving the user code and data, but finding & gathering up all the kernel data structures relating to the process that is leaving.
     - E.g. it may have open files, running timers, queued incoming messages, and other bits and pieces of information scattered around the kernel.
     - These must all be carefully removed from the source machine and successfully reinstalled on the destination machine.
     - There are no theoretical problems here, but the practical engineering difficulties are substantial.

- When the process is gone, it should leave the machine in the same state in which it found it, to avoid disturbing the owner.
  - This requirement means that not only must the process go, but also all its children and their children.
  - Network connections, and other system-wide data structures must be deleted, and some provision must be made to ignore RPC replies and other messages that arrive for the process after it is gone.
  - Temporary files must be deleted, and if possible, any files that had to be removed from its cache restored.

Particular case: Volunteer computing

- BOINC - standard
  - Open-source software for volunteer computing
  - Use the idle time on computers (Windows, Mac, or Linux) for many types of scientific research.
  - Appls: diseases, study global warming, discover pulsars
  - Current applications:
    - Astronomy/Physics/Chemistry:
      - Einstein@home, Milkyway@home, Quantum Monte Carlo@Home, Spinhenge@home,LHC@home, SETI@home, Cosmology@Home,uFluids@home
    - Biology and Medicine:
      - Rosetta@home, GPUGrid.net, Superlink@Technion, POEM@HOME, Malariacontrol.net, Docking@Home
    - Mathematics, computing, and games
      - Rectilinear Crossing Number, NFS@home, VTU@home, SHA-1 Collision Search, ABC@home, AQUA@home, PrimeGrid, Chess960@home, NQueens@home
- Others:
  - XtreemWeb
  - Desktop Grid
  - GridRepublic + Intel programme Progress Thru Processors

Processor Pool Model (Cluster)

- Although using idle workstations adds a little computing power to the system, it does not address a more fundamental issue:
  - What happens when it is feasible to provide 10 or 100 times as many CPUs as there are active users?

1. One solution, as we saw, is to give everyone a personal multiprocessor. However this is a somewhat inefficient design.
2. An alternative approach is to construct a *processor pool*, a rack full of CPUs in the machine room, which can be dynamically allocated to users on demand.
   - Instead of giving users personal workstations, in this model they are given high-performance graphics terminals.

- Conceptually, it is much closer to traditional timesharing than to the personal computer model, although it is built with modem technology.

Motivation

- If the file system can be centralized in a small number of file servers to gain economies of scale, it should be possible to do the same thing for *compute servers*.
- By putting all the CPUs in a big rack in the machine room, power supply and other packaging costs can be reduced, giving more computing power for a given amount of money.
- The model also allows for easy incremental growth.
  - If the computing load increases by 10 percent, you can just buy 10 percent more processors and put them in the pool.
- All the computing power is converted into "idle workstations" that can be accessed dynamically.
  - Users can be assigned as many CPUs as they need for short periods, after which they are returned to the pool so that other users can have them.
  - There is no concept of ownership here: all the processors belong equally to everyone.

The nature of the workload

- If all people are doing is simple editing and occasionally sending an electronic mail message or two => having a personal workstation is probably enough.
- If, on the other hand, the users
  - are engaged in a large software development project, frequently running make on large directories, or
  - are trying to invert massive sparse matrices, or
  - do major simulations or run big artificial intelligence or VLSI routing programs

  => constantly hunting for substantial numbers of idle workstations will be no fun at all.

  => In all these situations, the processor pool idea is fundamentally much simpler and more attractive.

Queueing System

- A queueing system is a situation in which users generate random requests for work from a server.
  - When the server is busy, the users queue for service and are processed in turn.
  - Common examples of queueing systems are bakeries, airport check-in counters, supermarket check-out counters, and numerous others.
- Queueing systems are useful because it is possible to model them analytically.

Examples of queueing systems

- Condor
  - a specialized workload management system for compute-intensive jobs
    - provides a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management.
  - full-featured batch systems
  - Users submit their serial or parallel jobs to Condor
    - Condor
      - places them into a queue
      - chooses when and where to run the jobs based upon a policy
      - carefully monitors their progress
      - ultimately informs the user upon completion
- LSF
  - JobScheduler is part of a workload management solutions
    - a single system image for a heterogeneous network of computers
    - dynamic and intelligent resource mapping and load balancing
    - centralized monitoring of resource load information and job information
    - calendar-driven/ event-driven / job scheduling
- PBS
  - operates on networked multi-platform UNIX environments

Hybrid model

- A possible compromise is to provide each user with a personal workstation and to have a processor pool in addition
  - Although this solution is more expensive than either a pure workstation model or a pure processor pool model, it combines the advantages of both of the others.
    - Interactive work can be done on workstations, giving guaranteed response
    - Idle workstations, however, are not utilized, making for a simpler system design: they are just left unused.
    - Instead, all non-interactive processes run on the processor pool, as does all heavy computing in general.
- This model provides fast interactive response, an efficient use of resources, and a simple design.
- Further readings: Grid computing, Cloud computing