

ITEE



What's New:

You should aim to catch up on previous tutorials; the last 2 will not introduce as much new material as before.

Learning objectives for this week:

- Understand the major concepts we've covered, including:
 - different types of communication
 - different approaches to distributing data and processing
 - networked file systems
- Apply these concepts to reasoning about distributed operating systems
- Apply these concepts to evaluating alternatives
- Understand issues in improving performance

Reasonably complete answers are supplied this time. Try to derive new questions from these.

1. Concepts

- a. When you use the world-wide web, are you using a distributed system, or a networked system? Do you see any difference between accessing a local versus a remote file?
A web browser works more like a networked system than a distributed system. Local pages are accessed with file: links while non-local pages are accessed with http: links. In a distributed system, local and remote resources are made to look as similar as possible.
- b. In a UNIX lab, when you access your home directory, are you using a purely local system, a networked system or a distributed file system?
Compare with the web page case: your home directory appears to be the same no matter where you are working (the local machine could be where you home directory is stored, but you don't need to know), which makes it a distributed system.
- c. It's common to use a point-to-point link in long-haul networks over very long distances, like across a big ocean. Why should this be the case?
There are very few links in this situation, and each may have different costs (e.g., satellite vs. undersea cable). Each service provider in a competitive situation may own its own link, and not be willing to share it, to maintain their competitive advantage.
- d. What are the advantages of stateless file servers?
Crash recovery is simpler: if either side goes down, since all transactions are treated as independent, there is no need e.g. for a server to try to work out after recovery which of its clients still need incomplete transactions, files to be kept open, etc. As a further question, consider disadvantages of a stateless file server.

2. Distributed Processing

- a. Consider each of the following cases and indicate in which cases you would consider data or process migration (neither or both may also be valid answers):
 - i. your application is small and rarely makes disk accesses, which are always small. It is running on a heavily loaded server, but could equally well run on your machine, but the files are on the server
In this case, there could be a case for process migration, if the file accesses are infrequent enough not to put a load on the network, and the cost of moving the process is low (small application). Because the disk accesses are small and rare, data migration is probably not a good idea, unless the file size is very small.
 - ii. your application is very large and rarely makes disk accesses, which are always small. It is running on a heavily loaded server, but could equally well run on your machine, but the files are on the server
This case makes it harder to argue for process migration because of the time to move the application. If the server is highly loaded and there is nothing else which could move, there could be a case.
 - iii. your application is very large and makes frequent large disk accesses. It is running on a heavily loaded server, but could equally well run on your machine, but the files are on the server
Process migration is harder to argue for here: this process appears to be best-suited to run on the server. Only if the server was not only very heavily loaded but no other application had a stronger case to move would process migration be an option here. Even so, the fact that the data is on the server makes process migration unattractive. It might be worth it in this case to migrate the data too, but only if it was clear that this solved a longer-term scheduling problem.
- b. It takes 10ms for any network transaction, plus the transfer time. The transfer time is size of data to move / networks speed. In each of the following, calculate the time saved (or extra) resulting from migrating the process:
 - i. size of process: 1Mbyte; time to run on busy server: 1s; time to run on client: 0.1s; 100Mbit/s network
*Time to migrate: $10ms + (1Mbyte / 100 Mbit/s) = 0.01s + 1/12.5s = 0.09s$
Time saving once moved: $1s - 0.1s = 0.9s$ – so worth moving*
 - ii. size of process: 1Mbyte; time to run on busy server: 1s; time to run on client: 0.1s; 1000Mbit/s network
*Time to migrate: $10ms + (1Mbyte / 1000 Mbit/s) = 0.01s + 1/125s = 0.018s$
Time saving once moved: $1s - 0.1s = 0.9s$ – so worth moving*
 - iii. size of process: 10Mbytes; time to run on busy server: 1s; time to run on client: 0.1s; 100 Mbit/s network
*Time to migrate: $10ms + (10Mbyte / 100 Mbit/s) = 0.01s + 10/12.5s = 0.81s$
Time saving once moved: $1s - 0.1s = 0.9s$ – so worth moving but not so convincing*
 - iv. size of process: 10Mbytes; time to run on busy server: 60s; time to run on client: 0.5s; 100Mbit/s network
*Time to migrate: $10ms + (10Mbyte / 100 Mbit/s) = 0.01s + 10/12.5s = 0.81s$
Time saving once moved: $60s - 0.5s = 59.5s$ – clearly worth moving*
 - v. size of process: 10Mbytes; time to run on busy server: 60s; time to run on client: 0.5s; 1000Mbit/s network
*Time to migrate: $10ms + (10Mbyte / 1000 Mbit/s) = 0.01s + 10/125s = 0.09s$
Time saving once moved: $60s - 0.5s = 59.5s$ – clearly worth moving*

- c. What can you say in general about process migration, based on (b), and any other good points you can think of?
These examples don't make the point as clearly as they could: with a small process, the 10ms start up overhead masks the improvement of going to a faster network. In general, if the process is small, unless the execution time gain is very significant (e.g., a processor is heavily overloaded), process migration is of questionable benefit. On the other hand, migrating a very large process can put a significant load on a network. The exact conditions in which process migration is worthwhile can be quite narrow, and rely on knowing how much longer a process will continue to run, which is not always known.
- d. Why, in general terms, is it hard to handle failures in distributed systems?
In a distributed system the cause and nature of the failure can be hard to determine. Has a response failed to arrive because the network has failed? Are messages just being delayed by congestion? Has a client or server crashed or suffered a software failure? Because the network is an inherently unreliable medium, it is much harder to know where the failure occurred and where to put the responsibility for recovery.

3. Distributed File Systems

- a. Which of the following file operations assume that a server must maintain client state?
- i. open or close a file
Both: otherwise the operations make no sense.
 - ii. allow a client to cache a file
Hard to implement without state: if the original file is altered, the cached copies need to be updated.
 - iii. allow a client to lock a file
Hard to see how this could be done without the server maintaining state – unless a group of cooperating clients maintain their own state, but in that case, the server isn't "allowing" the clients to lock the file.
 - iv. allow a client to read a file, starting from a specified location
If the location is specified as part of the operation, no state needs to be maintained: all the needed information is supplied at once.
 - v. allow a client to read a file, starting from where it left off
Here state needs to be maintained otherwise the server doesn't know where the last read ended.
- b. In general terms, explain why stateful file servers are becoming the norm, despite the theoretical advantages of stateless servers.
Performance is the biggest issue: client-side caching is hard to implement (see (a(ii))) without stateful servers. Also, locking and any operations which require files to be opened specifically (e.g. exclusive write) need state on the server. While these problems can be worked around, in the end, solving the problems of stateful servers seems to be easier.
- c. Does NFS provide location transparency or location independence? Explain.
Only the first – the latter requires that it be possible to move a file without changing its name, and NFS doesn't have that capability. See definitions in the book [p 575].
- d. What kind of file replication is used in the world-wide web? Consider implementations of clients, servers, and other performance-enhancing services.
Client-side caching is implemented at 2 levels: client machine's browser caches, and proxy caches at the user's site. ISPs may also implement transparent caching. Servers may also use caches, and there are services like Akamai, who automatically replicate services on demand around the world. Servers on large sites are often configured to redirect traffic to a server farm (you can sometimes see this in a rewritten URL). Try contacting a variety of sites, and see if you can work out which of these or other mechanisms are being used.