A comparison of two distributed systems

Finny Varghese

Topics

- Design Philosophies
  - Application environment
  - Processor allocation
- Design Consequences
  - Kernal Architecture
  - Communication Mechanism
  - File system
  - Process Management

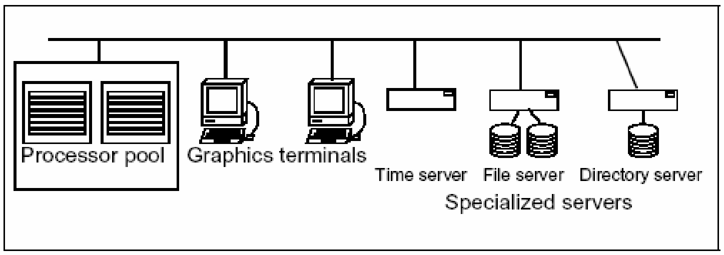
Amoeba vs. Sprite

- 2 philosophical grounds
  - Distributed computing model vs. Unix-style applications
  - Workstation-centered model vs. combination of terminal with a shared processor pool

Amoeba vs. Sprite

Amoeba	Sprite
user level IPC mechanism	RPC model – Kernal use
Caches files only on servers	Client-level caching
Centralized server – to allocate processors	Process migration model

Amoeba System



Sprite System

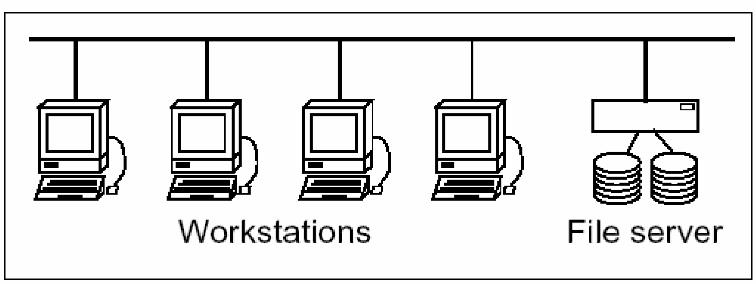


Figure 2: A Sprite system consists of workstations and file servers.

Design Philosophies

1. How to design a distributed file system with secondary storage shared?

2. How to allow collection of processors to be exploited by individual users

Application Environment

Amoeba

- Process or file = obj
  - *Capability*
    - *Port* – hides the server from objects
- Uniform communication model
- Easier - writing distributed application
- Orca – programming language

Sprite

- Eases – transition from time-sharing to networked workstations
- Caching file data – on workstations
- Little or no IPC

Processor Allocation

- Pure "workstation" – execute tasks on one machine
- Pure "processor pool" – equal access to all processors

- Amoeba – closer to processor pool
- Sprite – closer to workstation model

Processor Allocation - Amoeba

- "pool processor" – network interface and RAM
- Unlike pure – processors allocation outside pool processors for system services
- Terminals – only display server

<u>3 reasons for this choice</u>

1. Assumption that processor & memory price decrease
2. Assumption that the cost of adding new processor would be less than adding workstation
3. Entire distributed system – as a time sharing system

Processor Allocation - Sprite

- Priority, processing power of a workstation
- Unlike pure workstations – uses processing power of idle hosts
- Dedicated file servers – not for applications

<u>3 reasons for this choice</u>

1. Isolate system load
2. Power of new machine – better interface
3. No difference between graphic terminal and diskless workstation, except for memory in workstations

Design Consequences

Amoeba

- Dynamic load balancing
- No client file caching

Sprite

- Caches files on workstation
- Process migration

Design Consequences
Kernal Architectures

- Amoeba – microkernel
  - Ex. *Time of day* clock – provided by network wide server
- Uniformity, modularity, extensibility
- Dis. - services from processes slower than if kernel
- Dis. – no file caching
- Adv. – swapping and paging increases performance
- If good for trivial problems and as good as monolithic for complicated problems, then it outweighs any disadvantages.
- Sprite – UNIX monolithic model
  - Ex. *time of day* clock – provided by workstation
- 2 reasons
  - Implications of microkernel unclear
  - Cooperation of kernel facilities

Design consequence
Communication Mechanism

- Whole system – collection of objects – uses RPC
- Explicit acknowledgement in RPC
- Lower latency
- Lower bandwidth
  - 814 Kbytes/sec
- Kernel to kernel communication – RPC
- Implicit acknowledgement in RPC
- Higher latency
- Higher bandwidth
  - Blast protocol for large RPCs
  - 820 Kbytes/sec

Design consequence
Communication Mechanism

Size (Bytes)	Kernel-level Latency (msec)	
	Amoeba	**Sprite**
0	1.1	1.9
16384	20.0	19.5
30000	36.0	——

(a)

Size (Bytes)	User-level Latency (msec)	
	Amoeba	**Sprite**
0	1.2	7.9
16384	21.0	33.5
30000	36.0	62.8

(b)

Design Consequences
File Systems

- Amoeba – single globally shared, location-transparent file system
- No caching
  - Allows transparency & fault tolerance
- Sprite – single globally shared, location-transparent file system
- Caches data on both client and server

Design Consequences
File Systems

1. Transparent replication of files and directory entries
2. Bullet server - simpler, but includes some restrictions
3. No caching on client
4. Less memory to maintain open files

1. No replication
2. Files are immutable
3. Allows client caching
4. More memory to maintain open files

Design Consequences
File Systems

Amoeba/Sprite Comparison

Operation		Delay (msec)			
		Amoeba		Sprite	
open-close	*foo*	7.2		9.7	
	a/b/c/foo	7.6		10.4	
read				CACHE	NOCACHE
	10 Kbytes	14.0		2.8	18.6
	100 Kbytes	123.0		21.7	167.4
create-delete		BULLET	BULLET/DIR	CACHE	NOCACHE
	no data	33.0	288.0	50.9	50.9
	10 Kbytes	86.0	312.0	67.1	84.9
	100 Kbytes	367.0	617.0	101.4	411.1

Design Consequences
Process Model

- Amoeba – simple and efficient process model
- Virtual memory
- No demand-paging or swapping
  - Better performance of user-level RPC
- Threads for structuring server
- New process on new processor
  - exec_file
  - Avoids need to copy state of creating process

- Sprite – identical to BSD Unix
- Supports demand paging
- New process execution – *fork*
  - Copy of file

Design Consequences
Process Model

Operation	Time (msec)	
	Amoeba	Sprite
Context switch	0.5	1.6
Thread creation	2.4	(12.5)
fork	(169.5)	13.6
Program invocation	58.0	71.6

Design Consequences
Processor Allocation

- Amoeba – assign processes to many processors transparently
- *Run server* – selects processor
- Process starts in a different processor

- Sprite – gives priority to user over one workstation and runs all processes there
- Use of idle hosts – migration
- *Migd* – keeps track of idle hosts
- Process starts in local hosts

Design Consequences
Processor Allocation

Operation	Time (msec)	
	Amoeba	Sprite
Local	58	72
Remote (specified)	84	116
Remote (unspecified)	95	131

Design Consequences
Processor Allocation Drawbacks

- No multiple parallel applications to cooperate
  - Time-share each process among processors
- Default to local execution
  - Overload a workstation
- Use another host only if idle

Amoeba Evolution

- Parallel application
- Group communication
- Distributed Shared memory
- Wide area transparent systems

Sprite Evolution

- Log-structured file systems
- Striping files
- Buffering techniques
- Reliability
- Mach interoperability

Conclusion

1. Microkernals are not inferior to monolithic kernels
2. Desirability of uniform communication model – via RPC interface
3. Sprite benefits from client caching
4. Shows the need for hybrid systems
5. Compatibility with Unix – better for Sprite

Amoeba/Sprite Comparison

Thank you!

Questions?