Exam and Term Project

- Exam
  - Exam, Monday, April 8, 2024
  - Time: 3:05pm – 5:05pm; Room XXxxx
  - Close book, Close note, no phone, calculator, or computer or any internet access
- Term Project
  - Presentation, Monday and Tuesday, April 22
  - Term project report, April 25, 2024
  - Progress report March 20

Guest Lecture After Spring Break

- Wednesday
  - Dr. Bogdan Nicolae
  - AI, Deep Learning, and HPC

The Rest of Chapter 7

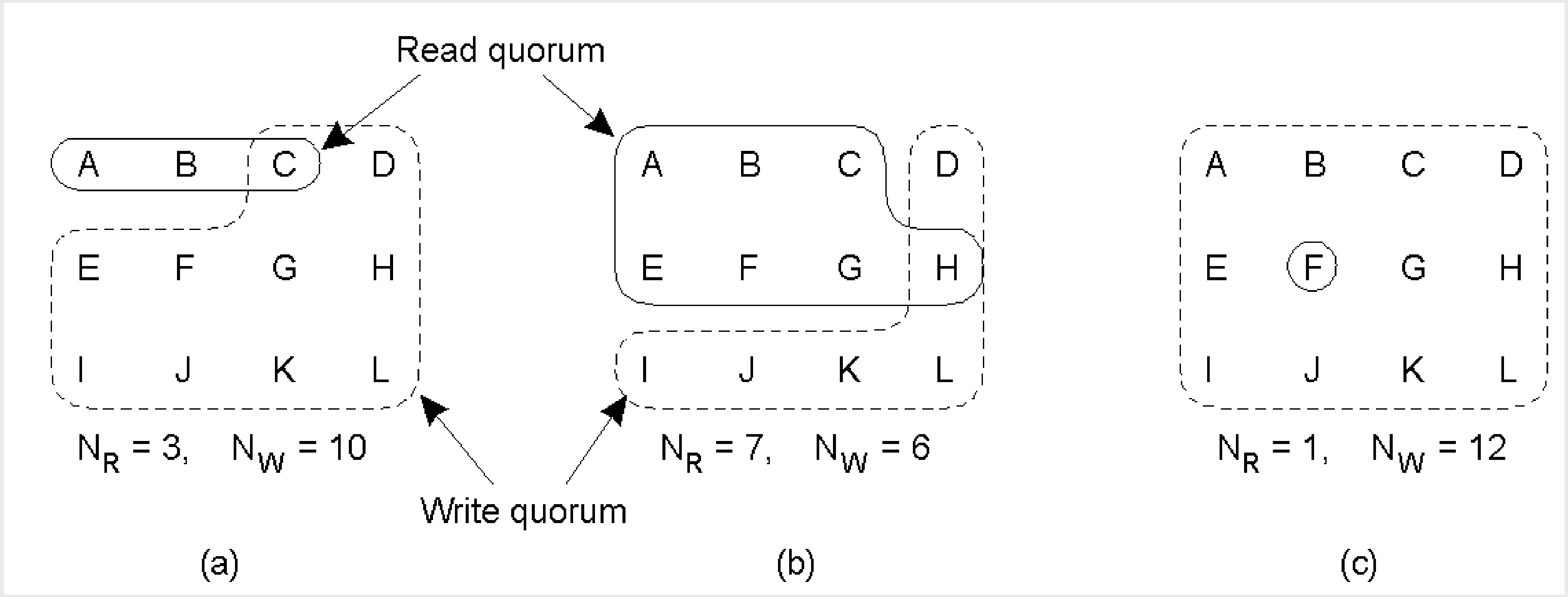
- Implementation issues (consistency protocols)
  - Primary-based
  - Replicated-write
  - Cache-coherence
- Replica Placement
- Putting it all together
  - Final thoughts

Replicated-write Protocols

- Relax the assumption of one primary
  - No primary, any replica is allowed to update
  - Consistency is more complex to achieve

- **Quorum-based protocols**
  - Use **voting** to request/acquire permissions from replicas
  - Example:
    - Consider a file replicated on N servers
    - **Update**: contact N/2+1 replicas and get them to agree to do the update (with a version number for the file)
    - **Read**: contact N/2+1 replicas and obtain the version number

Gifford's Quorum-Based Protocol



Two constraints: $N_R + N_w > N$; $N_w > N/2$

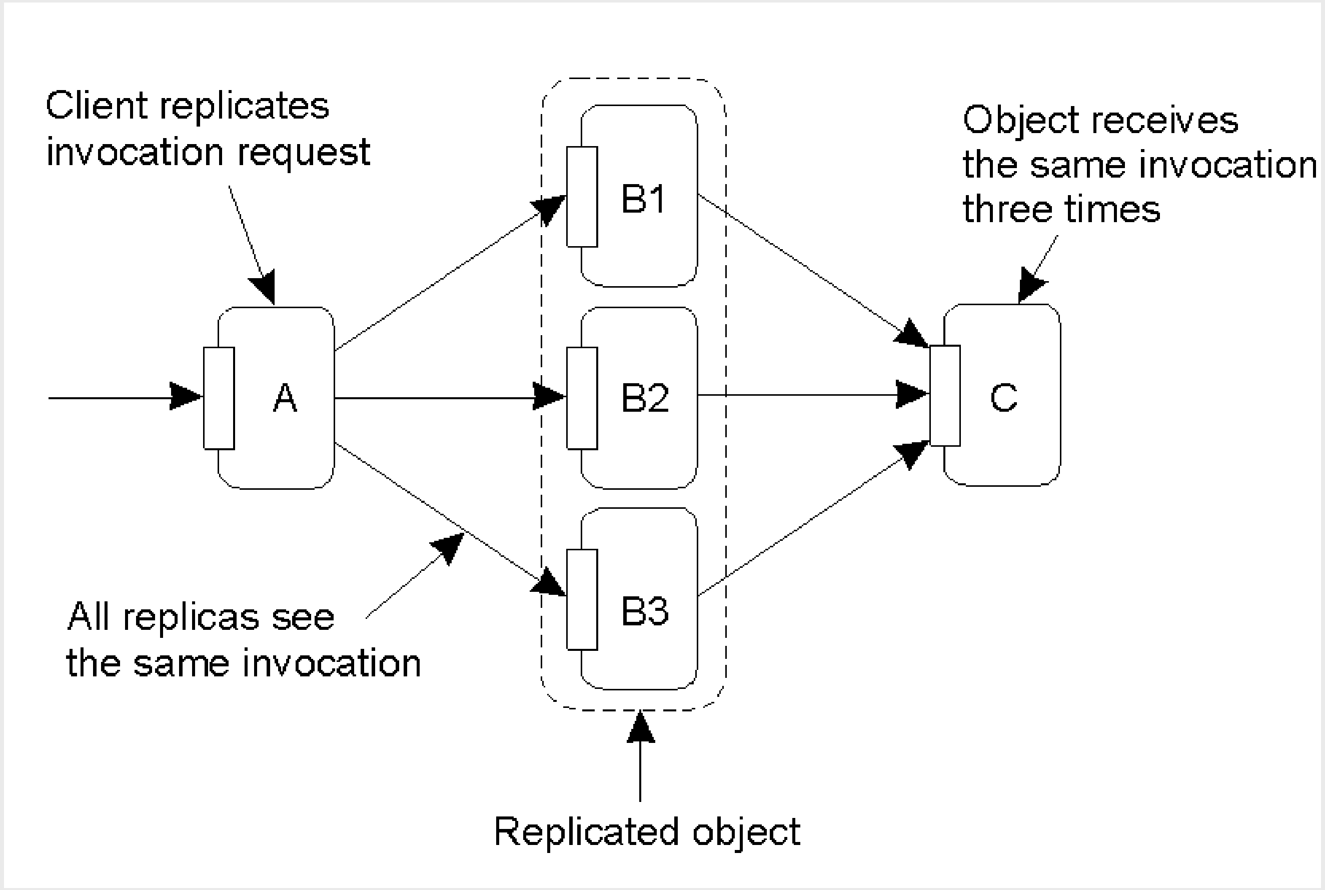
Impl. Issues of Consistency Model

- Passive Replication (Primary-Backup Organization)
  - Single primary replica manager at any time and one or more secondary replica manager
  - Writes can be carried out only the primary copy
- Active Replication:
  - There are multiple replica managers and writes can be carried out at any replica
- Cache Coherence Protocols
  - Client-centric consistency

Active Replication

- In active replication, each replica has an associated process that carries out update ops.
  - Client makes a request (via a front-end) and the request is multicast to the group of replica managers.
  - Totally-Ordered Reliable Multicast, based on Lamport timestamps
  - Implement *Sequential Consistency*
- Alternative is based on central coordinator (Sequencer)
  - First forward each op to the sequencer for a unique sequence number
  - Then forward the op, together with the number, to all replicas
- Another problem is replicated invocations
  - Object A invokes object B, which in turn invokes object c; If object B is replicated, each replica will invoke C independently, in principle
  - This problem occur in any client-server problem
  - No satisfactory solutions

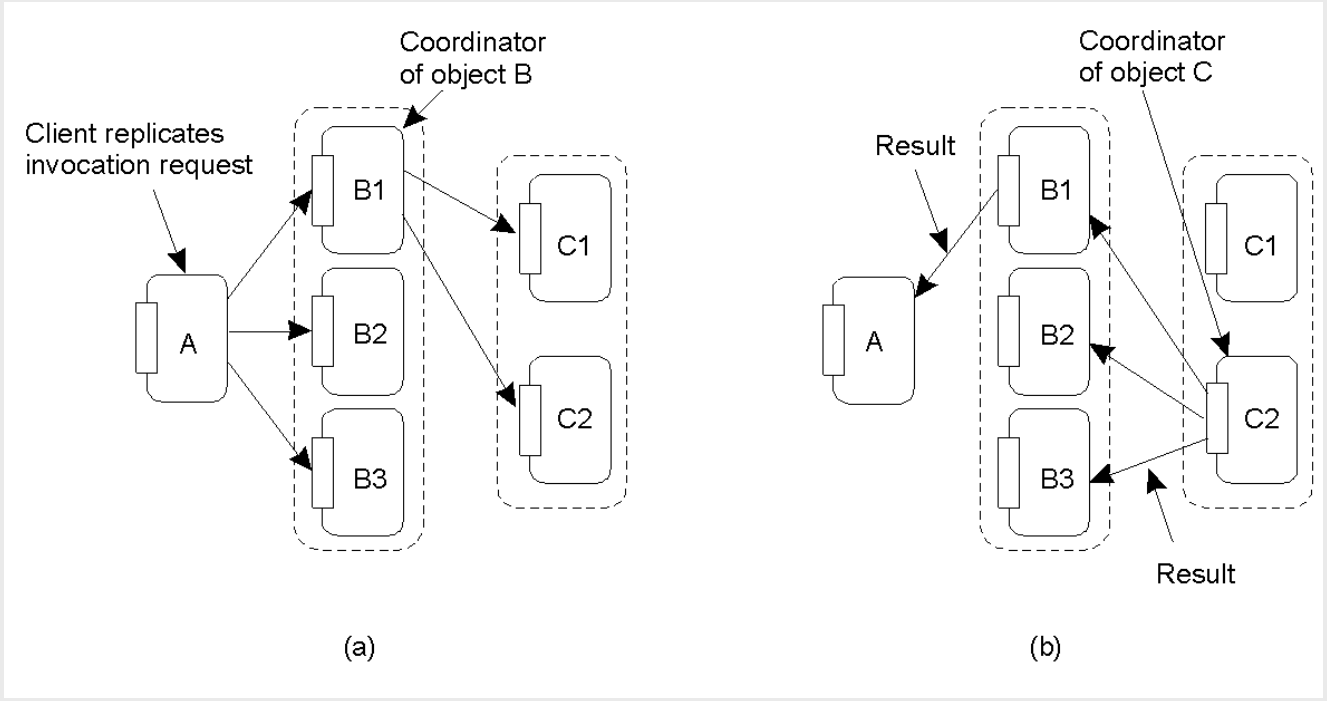
Active Replication


Client replicates invocation request
A
B1
B2
B3
C
Object receives the same invocation three times
All replicas see the same invocation
Replicated object

- Totally-ordered multicast, sequencer
- The problem of replicated invocations.

Replica-aware Active Replication

a) Forwarding an invocation request from a replicated object.
b) Returning a reply to a replicated object.

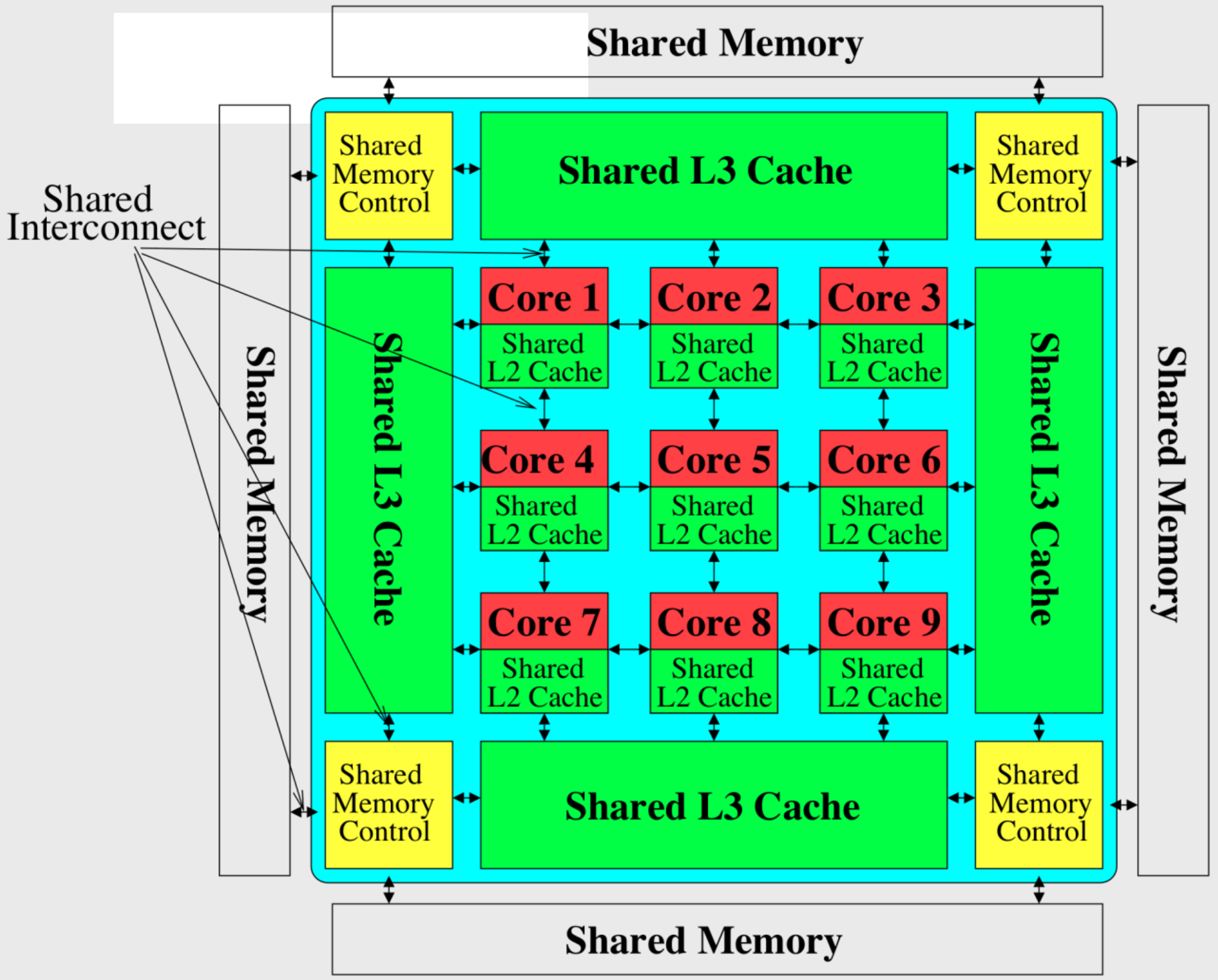


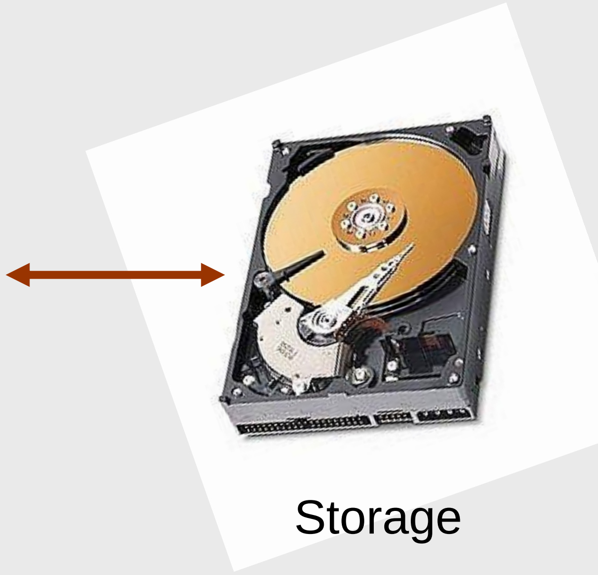
Solution: Middleware, unique identifier
Forwarding an invocation request from a replicated object.
Returning a reply to a replicated object

Impl. Issues of Consistency Model

- Passive Replication (Primary-Backup Organization)
  - Single primary replica manager at any time and one or more secondary replica manager
  - Writes can be carried out only the primary copy
- Active Replication:
  - There are multiple replica managers and writes can be carried out at any replica
- Cache Coherence Protocols
  - Client-centric consistency

Memory System: A ***Shared Resource*** View



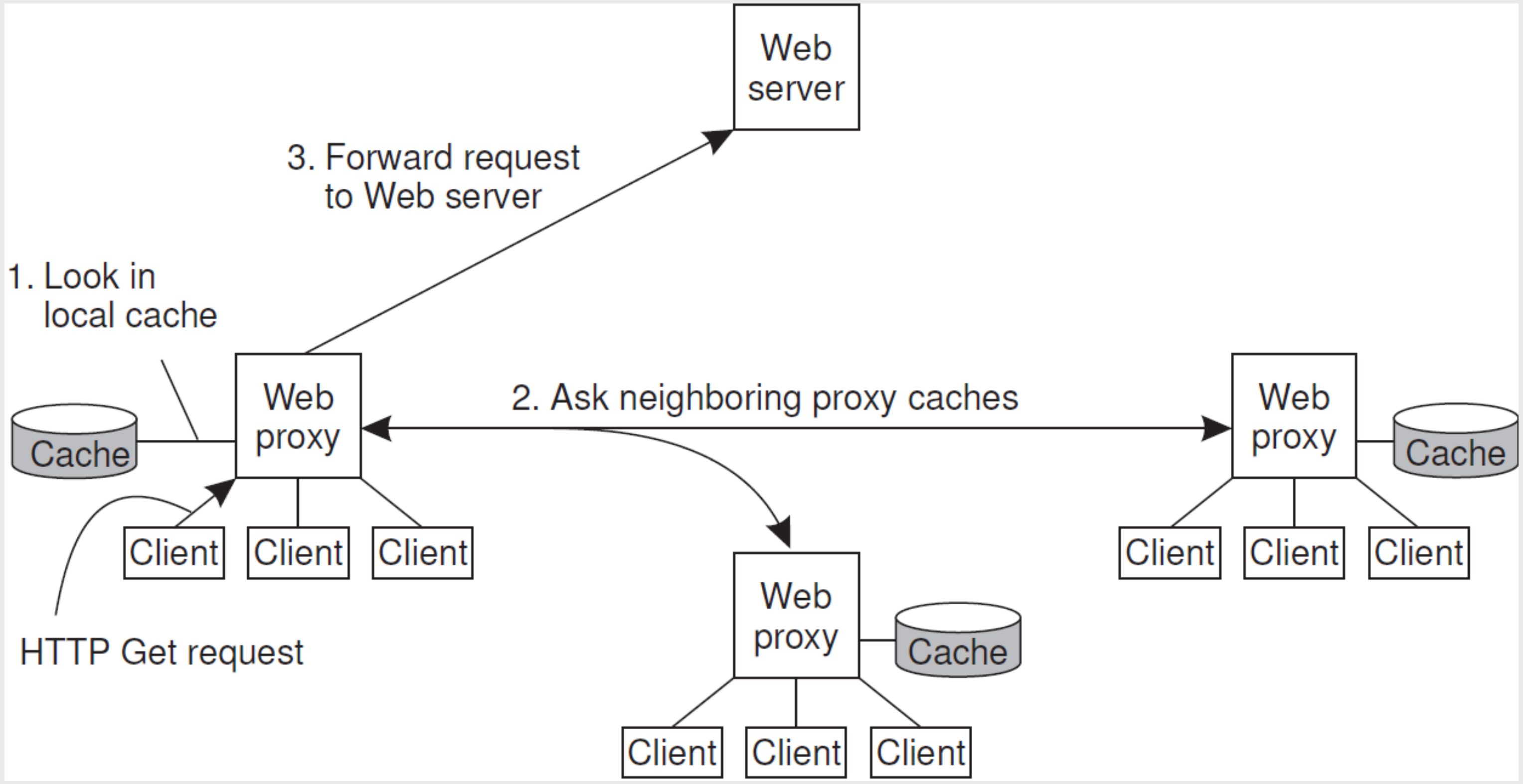


Most of the system use shared bus for moving data

Cache-coherent Protocols

- Mostly used for shared-memory systems
  - Based on hardware support (snooping or broadcast) or software-based solutions
- Two major design issues:
  - Coherence enforcement strategy
    - How to enforce the consistency?
    - Central control, sending invalidation or update
    - Write-through and write-back caches
  - Coherence detection strategy
    - When to detect the inconsistency?
    - Distributed database: During, Optimistic, Committed

Cooperative Caching Replication (in the Web)



- Shared cache

The Rest of Chapter 7

- Implementation issues (consistency protocols)
  - Primary-based
  - Replicated-write
  - Cache-coherence
  - Epidemic protocols
- Putting it all together
  - Final thoughts
- Replica placement

Epidemic Protocols

- Used in Bayou system from Xerox PARC
- Bayou: weakly connected replicas
  - Useful in mobile computing (mobile laptops)
  - Useful in wide area distributed databases (weak connectivity)
- Based on theory of epidemics *(spreading infectious diseases)*
  - Upon an update, try to "infect" other replicas as quickly as possible
  - Pair-wise exchange of updates (*like pair-wise spreading of a disease)*
  - Terminology:
    - Infective store: holds an update and willing to spread to others
    - Susceptible store: not yet been updated
    - Removed store: not willing or able to spread its update

Spreading an Epidemic

- Anti-entropy
  - Step: random find the partner
  - Push, pull, and updates to each other
  - Claim: A pure push-based approach does not help spread updates quickly (Why?)

- Rumor spreading (aka *gossiping*)
  - Step: end spreading if the listener already know the news

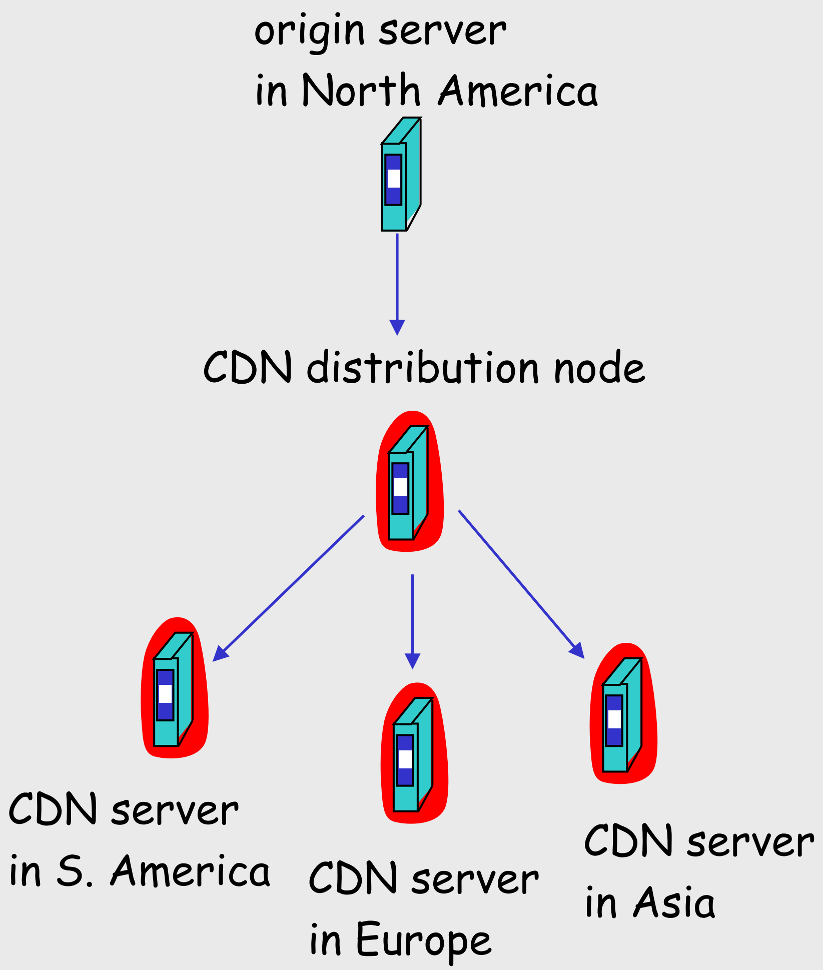
  - Con: no guarantee

Removing Data

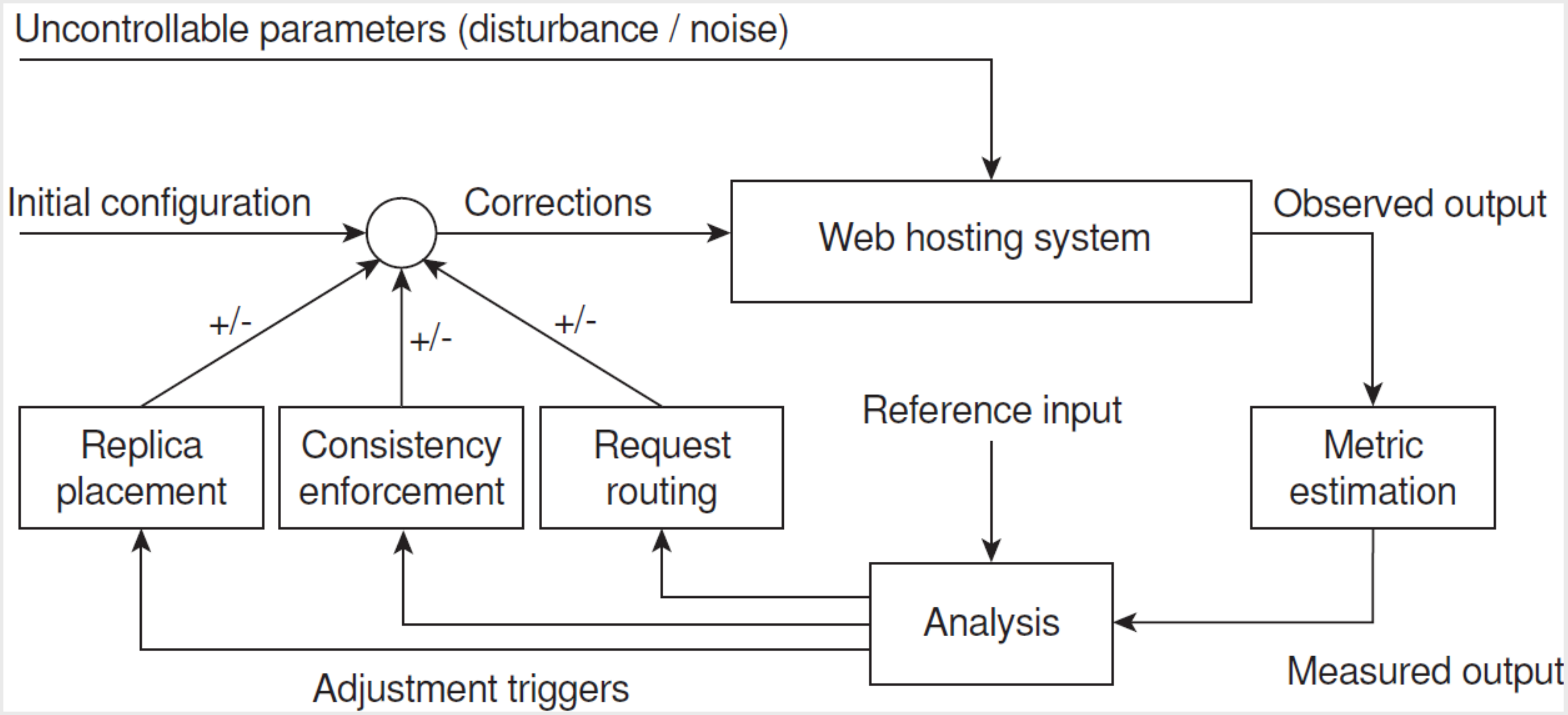
- Deletion of data items is hard in epidemic protocols
- Example: server deletes data item *x*
  - No state information is preserved
    - Can’t distinguish between a deleted copy and new copy!
  - Death certificate, with time

Replica Placement: server-initiated

- Dynamic replica placement of replica is a key to content delivery network (CDN).
  - CDN company installs hundreds of CDN servers throughout Internet
  - CDN replicates its customers' content in CDN servers. When provider updates content, CDN updates servers
- Key issue: When and where replicas should be created or deleted

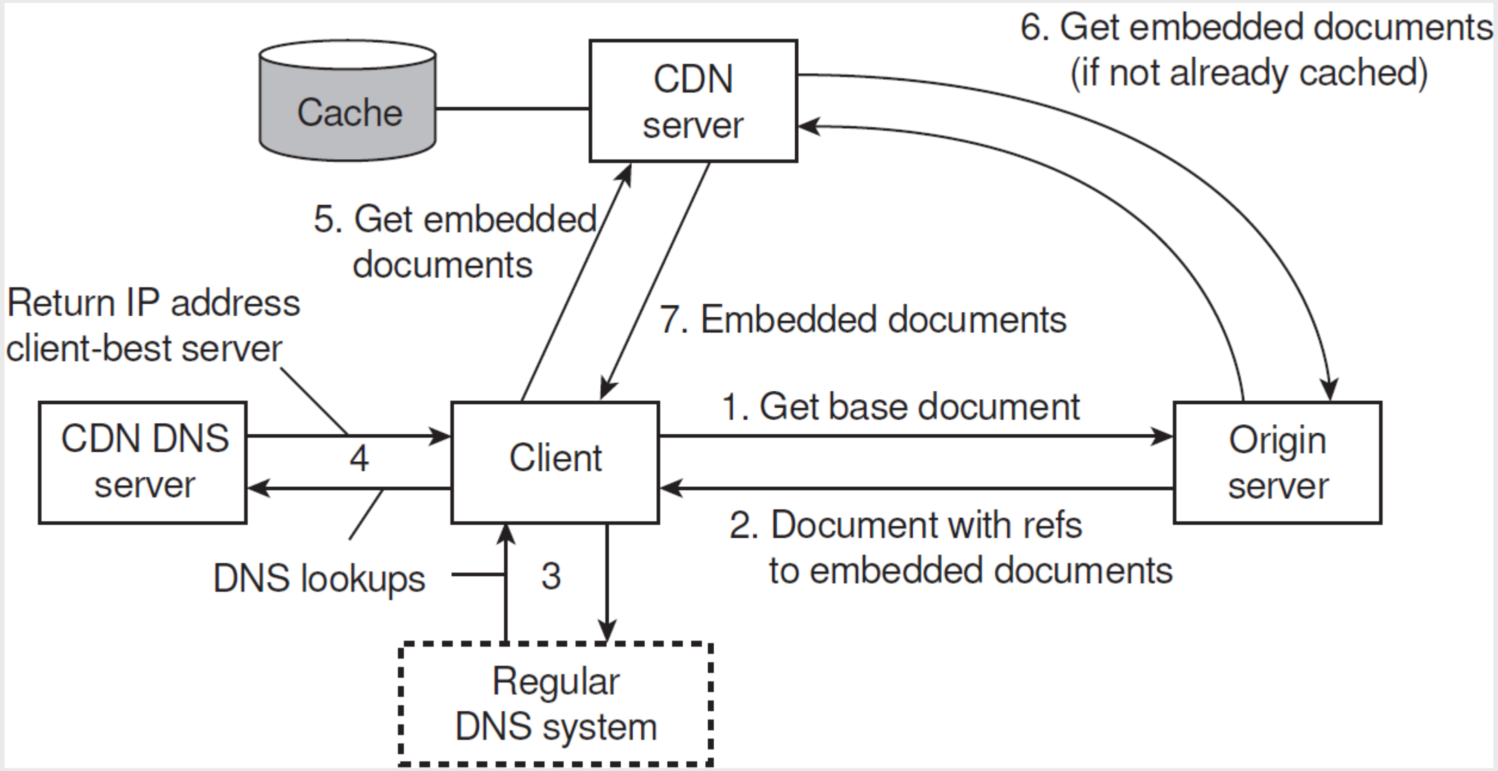


Content Delivery Network (CDN) as a Feedback-Control System



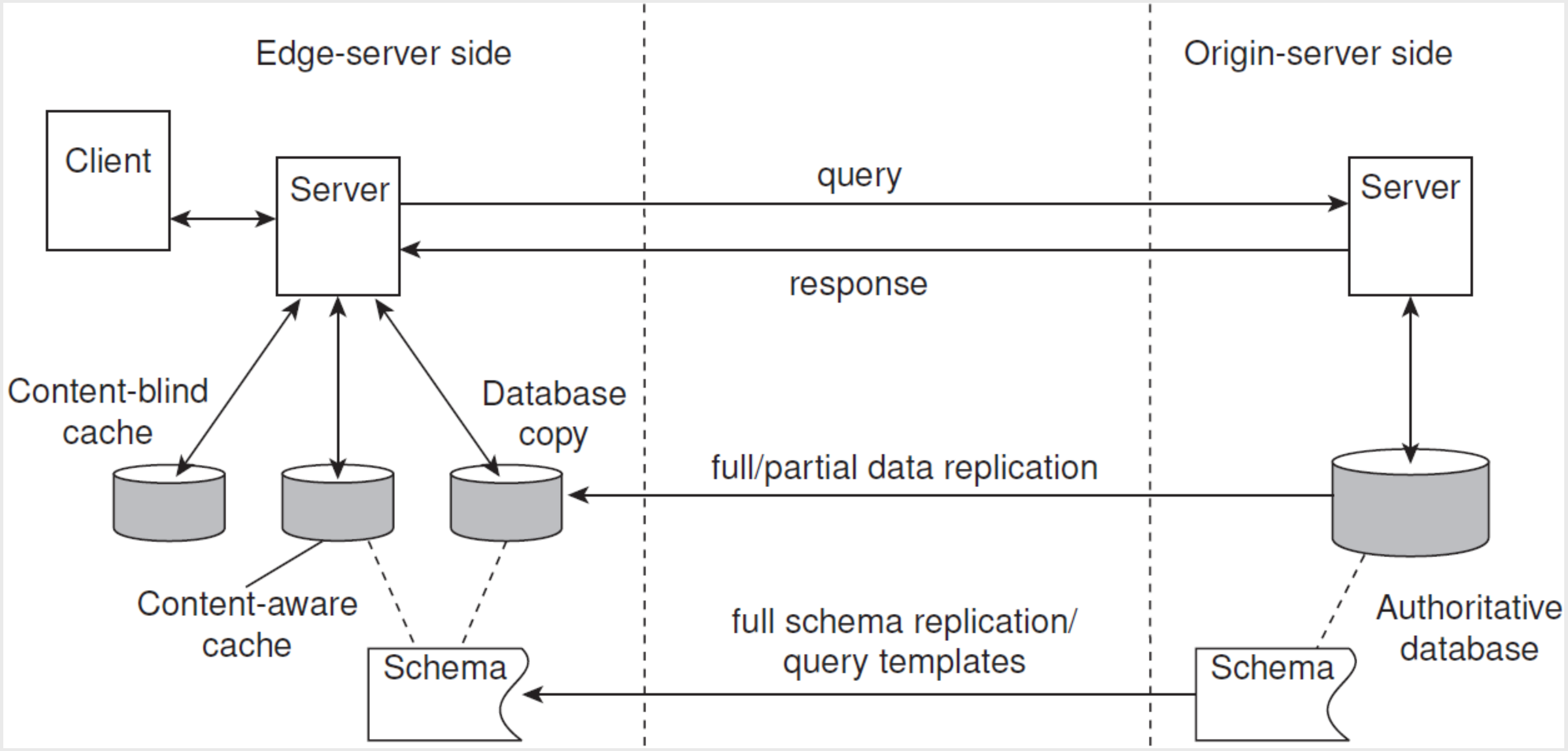
- Metric estimation, Adaptation triggering, Taking appropriate measures

Akamai CDN in Principal


6. Get embedded documents (if not already cached)
Cache
CDN server
5. Get embedded documents
Return IP address client-best server
7. Embedded documents
1. Get base document
CDN DNS server
4
Client
Origin server
2. Document with refs to embedded documents
DNS lookups
3
Regular DNS system

- TCP handoff, DNS redirection, HTTP redirection
- DNS: transparent
- HTTP: Substitute (replace) URL

Edge Server Approach in Web Applications



- Partial replication
- Content-aware cache
- Content-blind cache

Final Thoughts

- Replication and caching improve performance and reliability in distributed systems
- Consistency of replicated data is crucial (but difficult to achieve)
  - Numerical deviation, Staleness deviation, ordering deviation
- Many consistency semantics (models) possible
  - Implementation overheads and complexity grows if stronger guarantees are desired
  - Need to pick an appropriate model depending on the application
    - Example: web caching: weak consistency is OK since humans are tolerant to stale information (can reload browser)
  - Data-centric vs client-centric
  - Propagate updates
  - Primary-based protocols and replicated-write protocols
- Readings: Chapter 7

Remarks

- Applicable to mobile computing
  - Mobile machine serves a primary server before disconnecting
  - While being disconnected, all update operations are carried out locally; other processes can still perform read operations
  - When connecting again, updates are propagated from the primary to the backups

In Summary

- Replication
  - data, file (web files), object replication
  - reliability and performance
- Consistency Models
  - Strict and Linearizability
  - Sequential Consistency
  - Relaxed Consistency
  - Entry Consistency
  - Client-Centric Model for Special Data Space
- Implementation Issues

Chapter 7 Continue

- Implementation issues (consistency protocols)
  - Primary-based
  - Replicated-write
  - Cache-coherence
- Putting it all together
  - Final thoughts
- Replica placement

Parallel Processing without consider Consistency

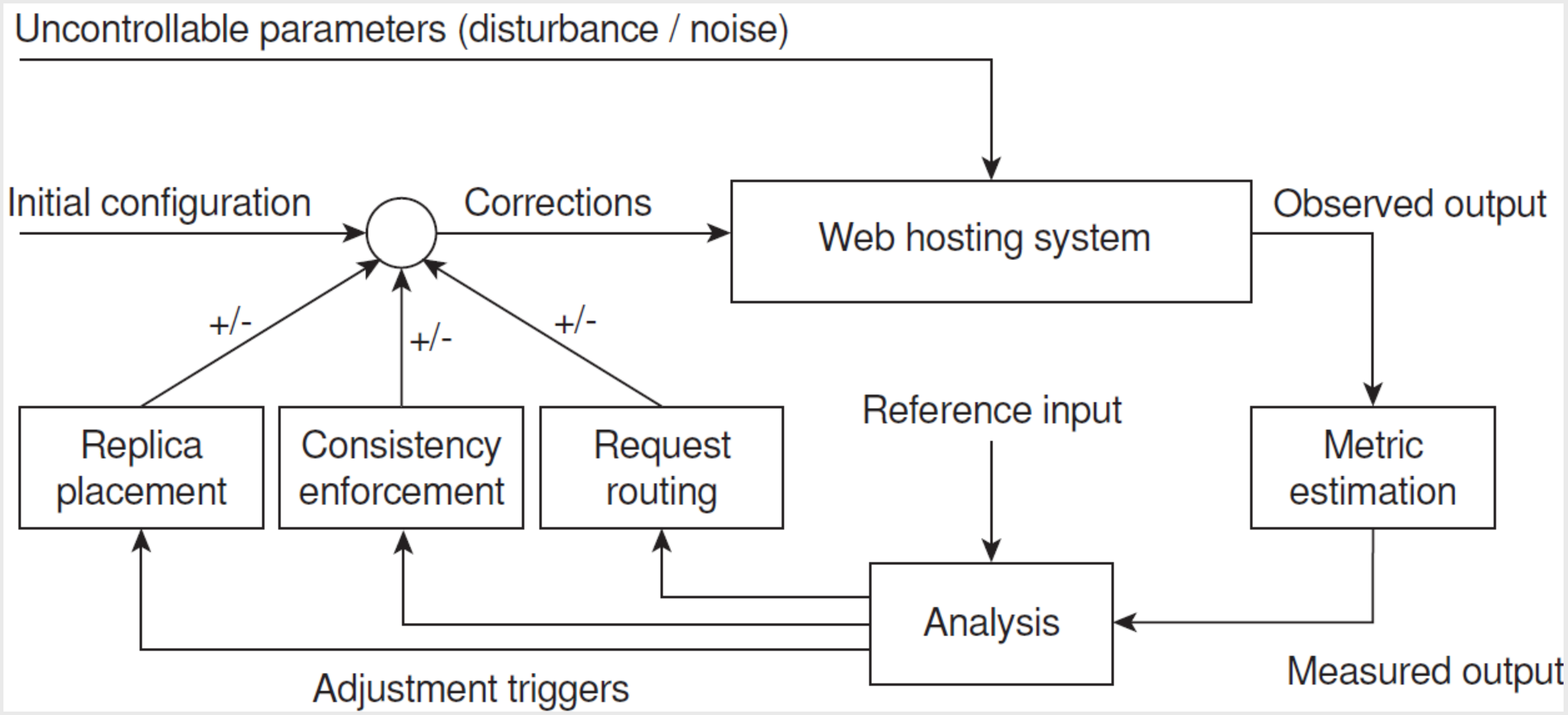
Distributed Scheduling: Motivation
(before web service)

- *Challenge*: multiple processing nodes=> scheduling not only is performed locally on each node but also globally across the system
- *Distributed scheduling* (aka load balancing): potentially useful for performance improvement or system utilization
- Example: a distributed system with *N* workstations
  - Model each w/s as identical, independent M/M/1 systems
  - Utilization *u*, P(system idle)=1-*u*
  - High utilization => little benefit
  - Low utilization => rarely job waiting
  - Probability high for moderate system utilization
    - Potential for performance improvement via load distribution

Design Issues

- The measure of load/resource availability
  - Must be easy to measure
  - Must reflect performance improvement
- Types of algorithms
  - Static: decision is made based on hardware
  - Dynamic: uses dynamic info, reschedule if necessary
  - Adaptive: policy varies according to load
- Types of task transfers
  - Preemptive versus non-preemptive
- Centralized versus decentralized
- What is the performance metric?
  - Mean response time, trustiness

Content Delivery Network (CDN) as a Feedback-Control System



- Metric estimation, Adaptation triggering, Taking appropriate measures

Types of Algorithms

- *Static algorithms*: Decisions are hard-coded into an algorithm with a priori knowledge of system.
- *Dynamic algorithms*: use system state information such as task queue length, processor utilization.
- *Dynamic rescheduling algorithms*: if system state has changed pass certain threshold reschedule.
- *Adaptive algorithms*: adapt the approach based on system state.
  - Dynamic algorithms collect load information from nodes even at very high system loads.
  - Load information collection itself can add load on the system as messages need to be exchanged.
  - Adaptive algorithms may stop collecting state information at high loads.

Types of Task Transfers

- *Preemptive task transfers*: transfer tasks that are partially executed (process migration).
  - Expensive as it involves collection of task states.
  - Task (process) state: virtual memory image, process control block, IO buffers, file pointers, timers, ...
- *Non-preemptive task transfers*: transfer tasks that have not begun execution.
  - Do not require transfer of task states.
  - Can be considered as task placements.
- Both transfers involve information on user's current working directory, task privileges/priority.

What is the different between task and process?

Algorithm Components

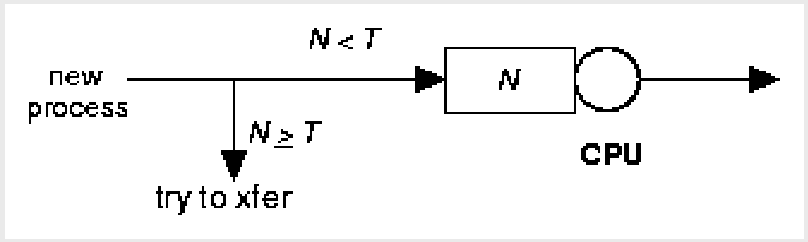
- *Transfer policy*: when to transfer a task?
  - Threshold-based policies are common and easy
- *Selection policy*: which task to transfer?
  - Prefer new task
  - Transfer cost should be small compared to execution cost
    - Select tasks with long execution times
- *Location policy*: where to transfer the task?
  - Polling, random, nearest neighbor
- *Information policy*: when and from where?
  - Demand driven [only if sender/receiver], time-driven [periodic], state-change-driven [send update if load changes]

Load Distributing Algorithms

- *Sender-initiated*:
  - distribution initiated by an overloaded node.
- *Receiver-initiated*:
  - Distribution initiated by lightly loaded nodes.
- *Symmetric*:
  - Initiated by both senders and receivers. Has advantages and disadvantages of both the approaches.

Sender-initiated Algorithm

- Transfer policy: use threshold



- Selection policy: newly arrived task
- Location policy: three variations
  - Random: transferred to a node at random.
  - Threshold: probe n nodes sequentially
    - Transfer to first node below threshold, if none, keep job
  - Shortest: poll N_p nodes in parallel
    - Choose least loaded node below T

Sender-initiated Algorithm

- Information Policy: demand-driven.
- Stability: can become unstable at high loads.
  - At high loads, it may become difficult for senders to find receivers.
  - Also, the number of senders increase at high system loads thereby increasing the polling activity.
  - Polling activity may make the system unstable at high loads.

Receiver-initiated Algorithm

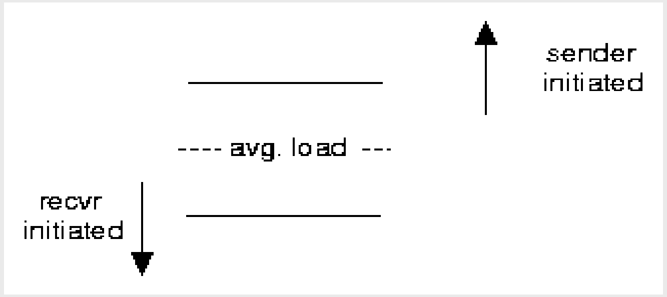
- Transfer policy: If departing task causes load $< T$, find a task from elsewhere
- Selection policy: newly arrived or partially executed task
- Location policy:
  - Threshold: probe up to N_p other nodes sequentially
    - Transfer from first one above threshold, if none, do nothing
  - Shortest: poll n nodes in parallel, choose node with heaviest load above T

Receiver-initiated Algorithm

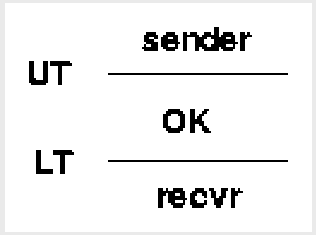
- Information policy: demand-driven.
- Stability: Not unstable since lightly loaded nodes initiate the algorithm
- Drawbacks:
  - Polling initiated by receiver implies that it is difficult to find senders with new tasks.
  - Effect: receiver-initiated approach might result in preemptive transfers. Hence transfer costs are more.
  - Sender-initiated: transfer costs are low as new jobs are transferred and so no need for transferring task states.

Symmetric Algorithm

- Nodes act as both senders and receivers: combine previous two policies without change
  - Use average load as threshold



- Improved symmetric policy: exploit polling information
  - Two thresholds: *LT*, *UT*, *LT* <= *UT*
  - Maintain sender, receiver and OK nodes using polling info
  - Sender: poll first node on receiver list …
  - Receiver: poll first node on sender list …



The GHS Solution

- Develop the Grid Harvest Service (GHS) system for task scheduling
- An adaptive measurement system, a set of dynamic scheduling algorithms, a sender-initiated trigger system for dynamic rescheduling
- A scheduler (hierarchical scheduling system) controls the adaptive measurement system, and make decision on where to migrate
- Assumption
  - Each task is a process
  - Resources are shared, with local and global task

- M. Wu and X.-H. Sun, "Grid Harvest Service: A Performance System of Grid Computing," Journal of Parallel and Distributed Computing, vol. 66, no. 10, pp. 1322-1337, 2006. (ACM Computing Review)
- M. Wu, X.-H. Sun, and Y. Chen, QoS Oriented Resource Reservation in Shared Environments, *in Proc. of 6th IEEE International Symposium on Cluster Computing and the Grid*, Singapore, May, 2006

Comments

- If a system rarely gets highly loaded, sender-initiated algorithms work better.
- High loads: receiver-initiated algorithms
- Widely fluctuating loads: symmetric algorithms
- Widely fluctuating loads + high migration cost for preemptive transfers: sender-initiated algorithms
- Need a scheduler which know global info, When
  - Migration cost is high
  - No information about the environment
  - Need coordinate communication or other activies

Chapter 8: Fault Tolerance

- Basic concepts in fault tolerance
- Masking failure by redundancy
- Process resilience
- Reliable communication
  - One-one communication
  - One-many communication
- Distributed commit
  - Two phase commit
  - Three phase commit
- Failure recovery
  - Checkpointing
  - Message logging