DISTRIBUTED RESEARCH ON EMERGING APPLICATIONS & MACHINES
dream-lab.in | **Indian Institute of Science**, Bangalore

DREAM:Lab

SE252:Lecture 11/12, Feb 10/19

IL03:Algorithms and Programming Patterns for Cloud Applications

Yogesh Simmhan

Summary of ILO 1

- Flynn's Taxonomy
- Shared vs Distributed Memory MIMD
- HPC, HTC, OLTP, Big Data
- Clusters, Grids, P2P, Clouds
- Task vs Data Parallelism
- Scalability
  - Amdahl, Gustafson, Strong, Weak

ILO 1

- Parallel and Distributed Systems Context
  - *Classify* and *describe* the architecture and taxonomy of
    - » parallel and distributed computing, ✓
    - » Shared and distributed memory, and ✓
    - » data and task parallel computing. ✓
  - *Explain* and *contrast* the role of Cloud computing within this space. ✓

DREAM:Lab

Lecture 11

ILO 3

- Algorithms and Programming Patterns for Cloud Applications
  - *Examine* the design of task and data parallel distributed algorithms for Clouds and
  - *use* them to construct Cloud applications.
  - *Demonstrate* the use of task graphs and Map-Reduce programming model.
  - *Apply* Amdahl's law and data locality principles to
  - *analyse* and *characterize* the potential speedup of Cloud applications.

Application & System Goals

- System owner
  - *Maximize* **resource utilization**
- Application owner
  - *Minimize* **Makespan** *for application*
  - *Minimize* Makespan *for workload*
  - *Minimize* **Cost**
- Makespan
  - End-to-end time taken for an application to be completed from the time it is submitted
- Weak vs Strong Scaling
- **Latency**
  - Time for application to start producing “useful” result

Application Analysis

- Decompose Makespan into constituents
  - *Job queuing time*
  - *Compute time (CPU)*
  - *I/O time (disk)*
  - *N/W time*
  - *System Overhead*
- Application Design
  - *Goal is to meet one of these **metrics** using scalable design patterns*

1) *Ab initio* vs. Retrofit

- Parallel Formulation
  - Introduce scalability into existing sequential application using concurrency
- Parallel Algorithm
  - Develop algorithm/application to specifically work on distributed systems

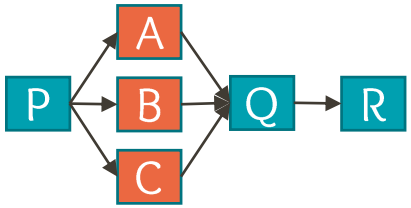
Application Decomposition

- Decompose monolithic application into discrete tasks

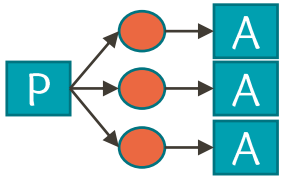
- Why?
  - Identify degrees of parallelism
    - » Divide and conquer
  - Identify dependencies, constraints
    - » Order of execution

2) Task vs Data Parallel

- Task Parallel
  - Perform different tasks at the same time…
  - On different processors/VMs
  - Limited by?
    - » Number of independent tasks



- Data Parallel
  - Operate on different data at the same time…
  - Limited by?
    - » Number of independent data "blocks"



- Hybrid is common – *MIMD*

Steps to Decomposition

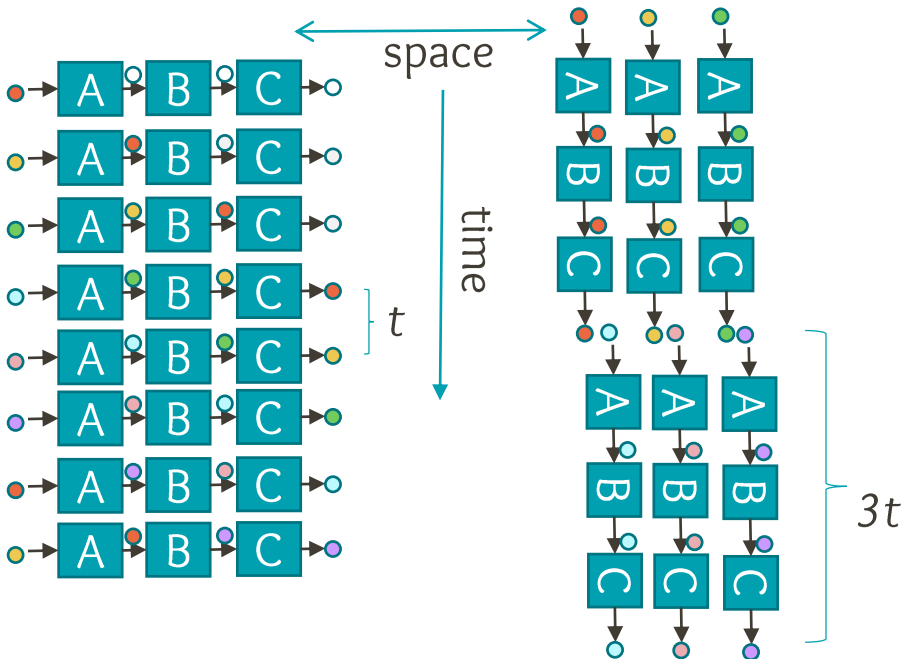
- Computation Decomposition/Partitioning:
  - Identify pieces of work that can be done concurrently
  - Assign tasks to different VMs
  - *Think Amdahl's Law…*
- Data Decomposition/Partitioning:
  - Decompose input, output & intermediate data across different VMs

Points to Consider

- Shared vs. Independent Data
  - Lock free, Write locks, Read locks
- Maximize *potential* speedup!
- Maximize concurrency but also reduce overheads
  - Impact on Strong vs Weak scaling

3) Pipelining

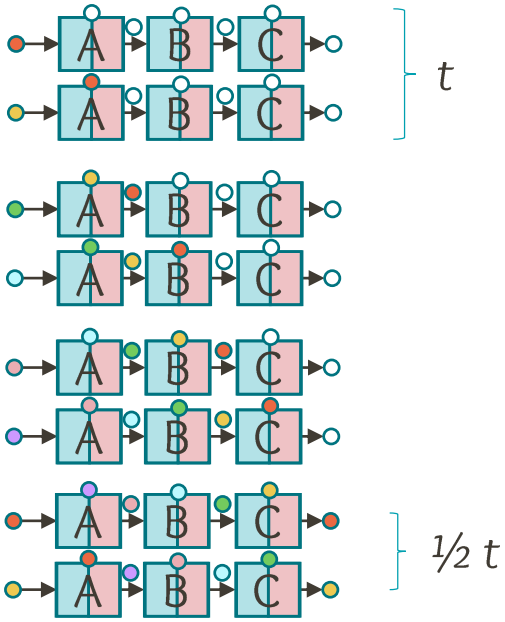
- Spread over time rather than space
  - May make scheduling easier
- Incremental, rather than coarse units of computation
  - Reduces time between results



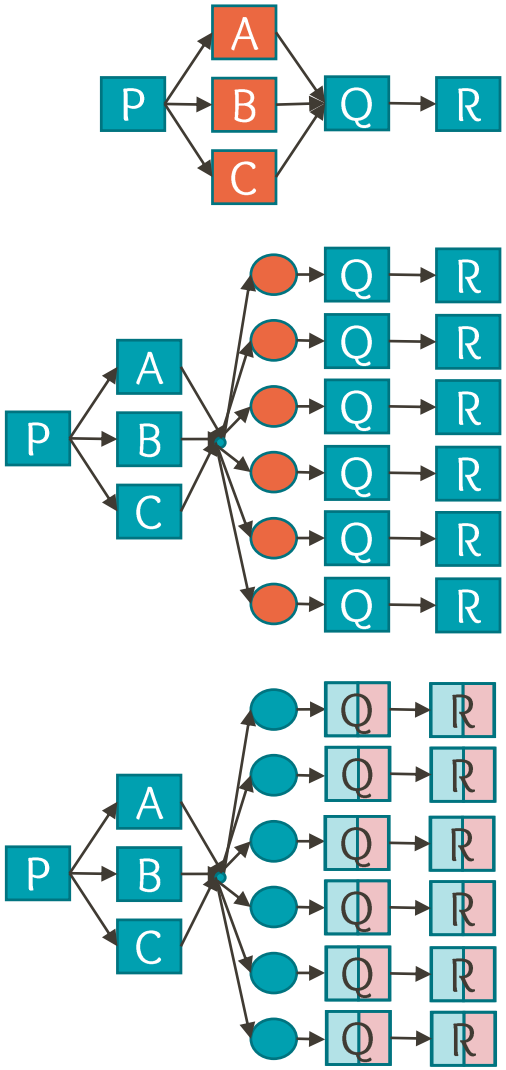
Chapter 3: Pipelining and Parallel Processing, Keshab K. Parhi, 1999

Pipelining

- Reduce the effective critical path thru' interleaving
  - Hide communication overheads. *Think VM data transfer.*
  - Makespan = I/O + Compute can be reduced to
  - Makespan = MAX(I/O, Compute)
- Reduce latency between tasks



Task/Data Parallel & Pipelining Orthogonal



- Helps increase degrees of parallelism
  - Reduce latency/makespan
- Improve resource utilization

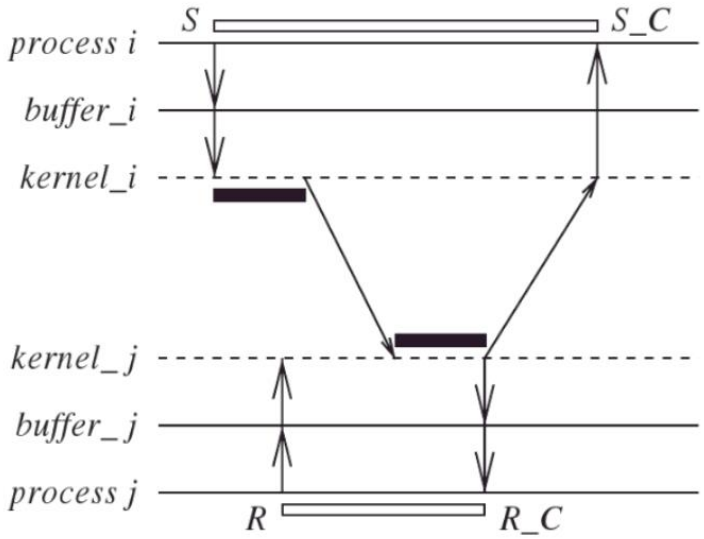
4) Synchronous vs Asynchronous

- Dependency between application sequence of ops, and how you handle it
- Synchronous vs Asynchronous
  - Single logical view vs. Interleaved view
  - Assembly line vs. On-demand
- Can be single or multi-threaded
- Synchronous seems natural, but
  - Asynchronous is more responsive
  - E.g. event driven programming

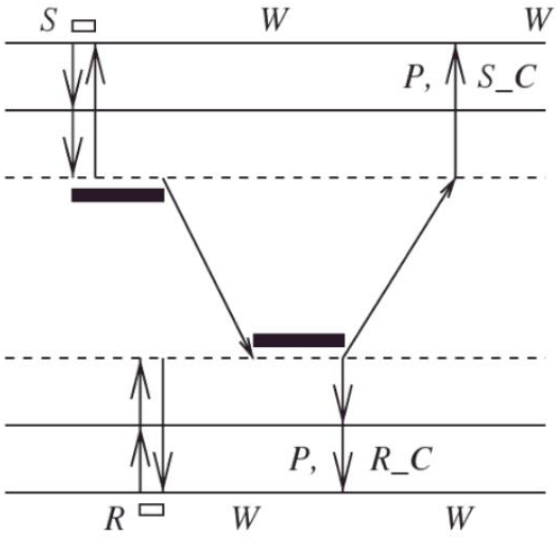
5) Blocking vs Non-blocking

- Blocking vs Non-blocking interaction between code
- Do you wait for completion or proceed with execution?
- *Related to but orthogonal to Sync/Async*

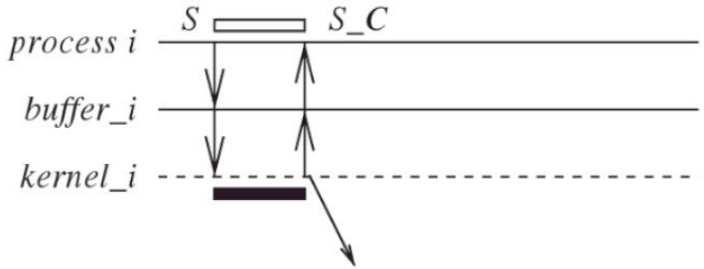
- Async, Blocking : Poll & Yield/Work
  - Suspend state & yield CPU/VM, or do other work
- Sync, Blocking: Wait/Sleep
  - Retain state & idle CPU/VM
- Async, Non-blocking: Call-back
  - Stateless/persisted/returned in call-back, yield CPU/VM
- Sync, Non-blocking: Independent, or…
  - Fire & forget



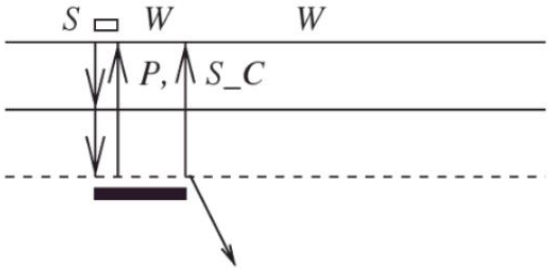
(a) Blocking sync. *Send*, blocking *Receive*



(b) Nonblocking sync. *Send*, nonblocking *Receive*



(c) Blocking async. *Send*



(d) Non-blocking async. *Send*

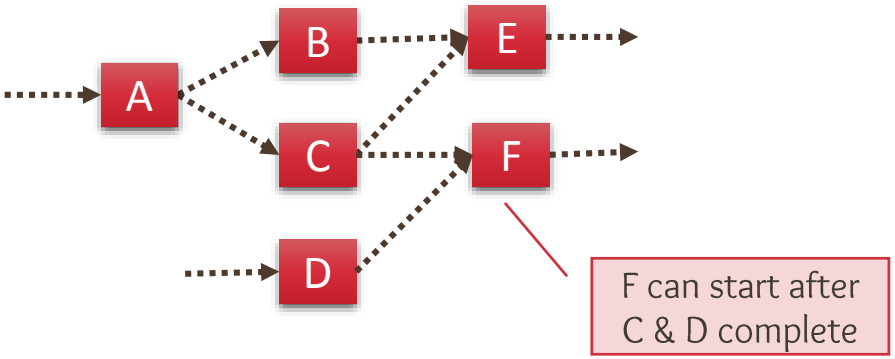
▬	Duration to copy data from or to user buffer		
▭	Duration in which the process issuing send or receive primitive is blocked		
S	*Send* primitive issued	S_C	processing for *Send* completes
R	*Receive* primitive issued	R_C	processing for *Receive* completes

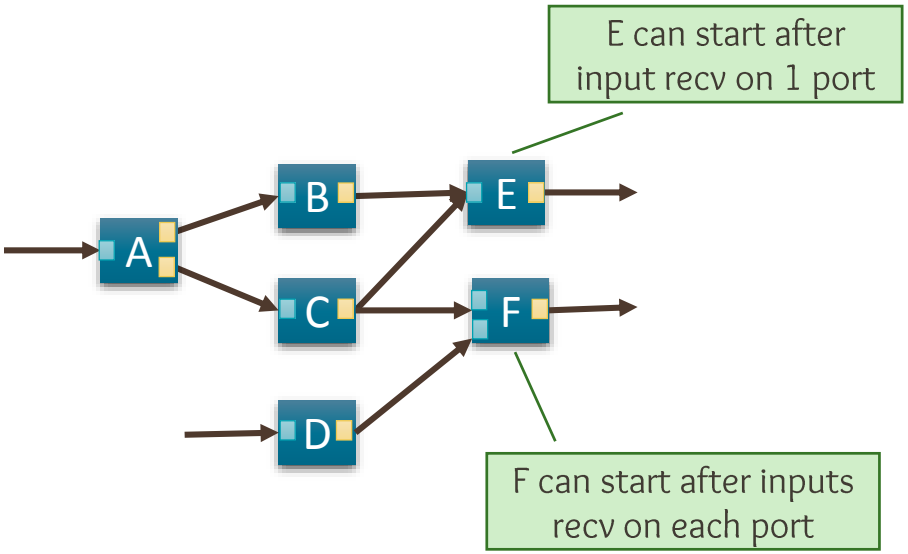
Distributed Computing: Principles, Algorithms, and Systems, Ajay D. Kshemkalyani, Mukesh Singhal,
Primitives for Distributed Communication, Figure 1.8

6) Task Graphs, Workflows

- Task graphs
  - Dependency between tasks that form an application
- Control dependency
  - Subsequent task cannot start till any/all previous task(s) completes
- Data dependency
  - Task cannot start till all inputs are available

Control vs Data Flow



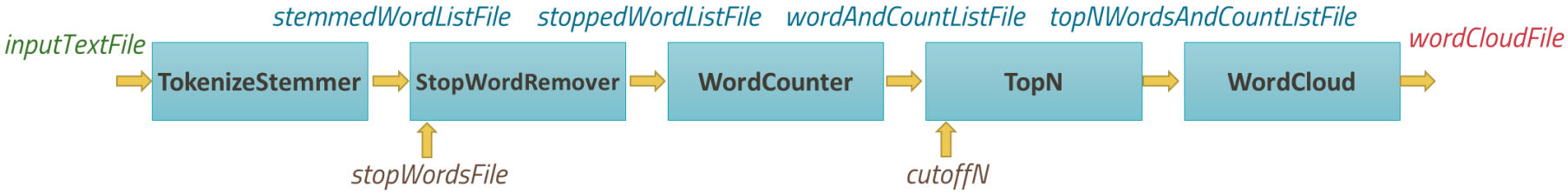


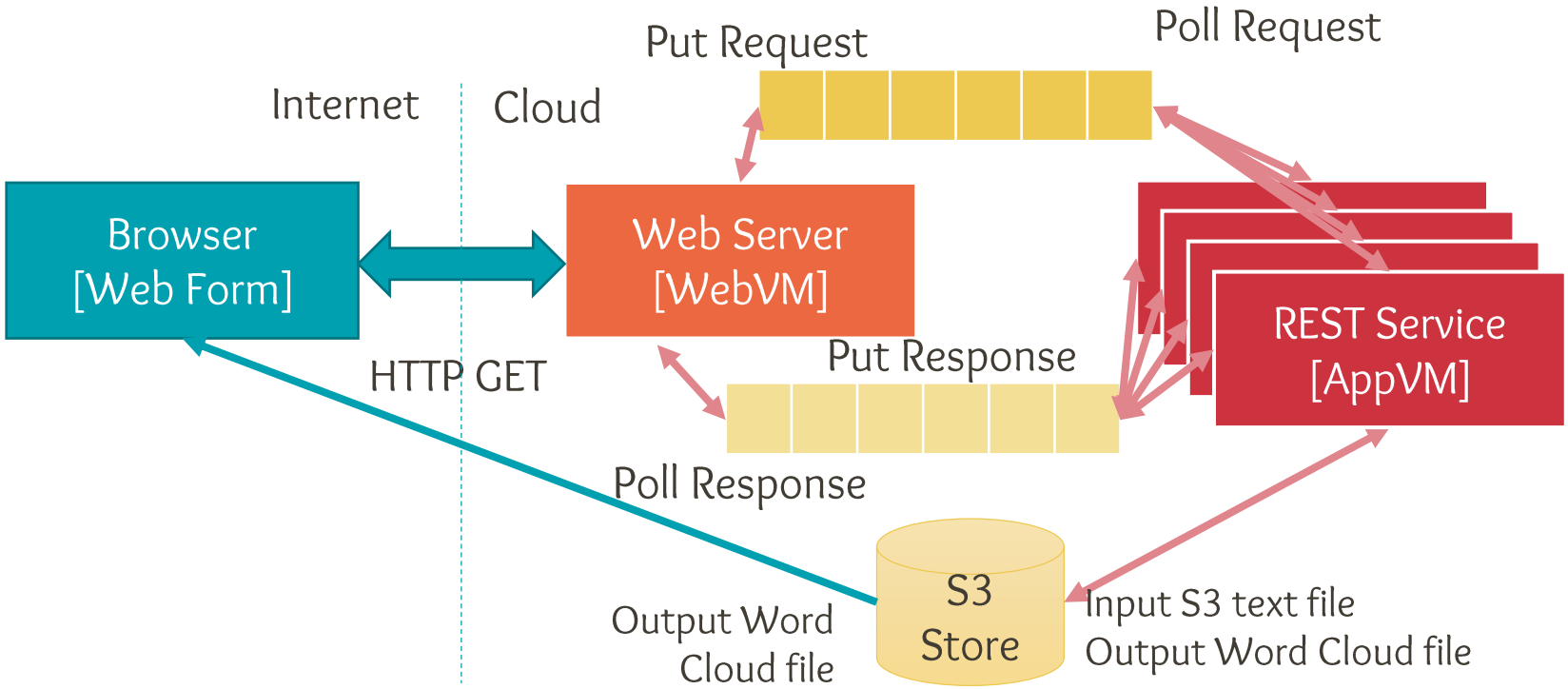
Dataflow has a *functional* model

7) Data Locality

- Hierarchy of costs
  - Network, Disk, Memory, Compute
- Network communication cost can be high in distributed systems
  - S3 → VM, VM →VM
- Concurrent disk I/O on same machine can be punitive
  - Cumulative disk I/O
- Attempt to reduce I/O transfers, Maximize bandwidth
- Temporal data locality
  - Motivates caching of data on local disk vs. S3
  - Issues?
- Spatial task locality
  - Move compute to data rather than data to compute

Simple Text Analytics Pipeline (SiTA)

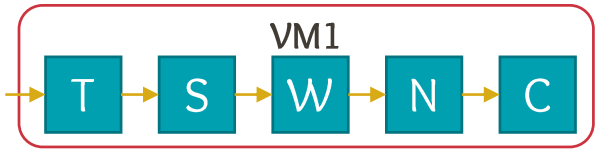




Applying Patterns to SiTA

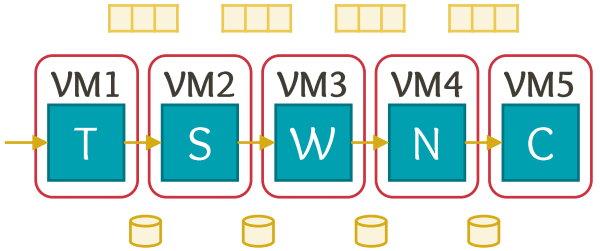
(1)

# of VMs: 1..6

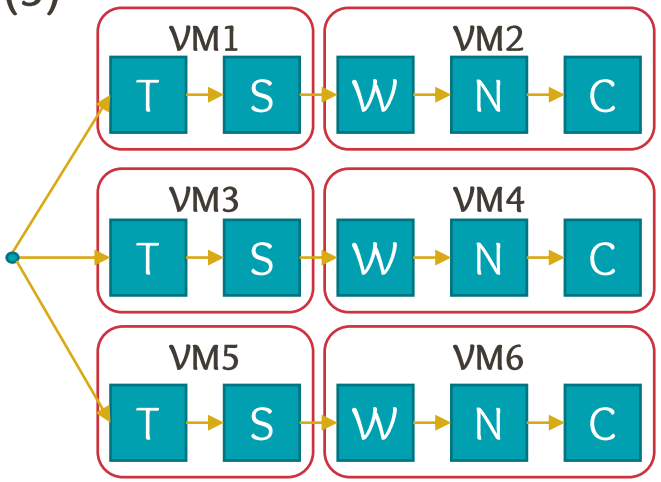


(2)

# of VMs: 5

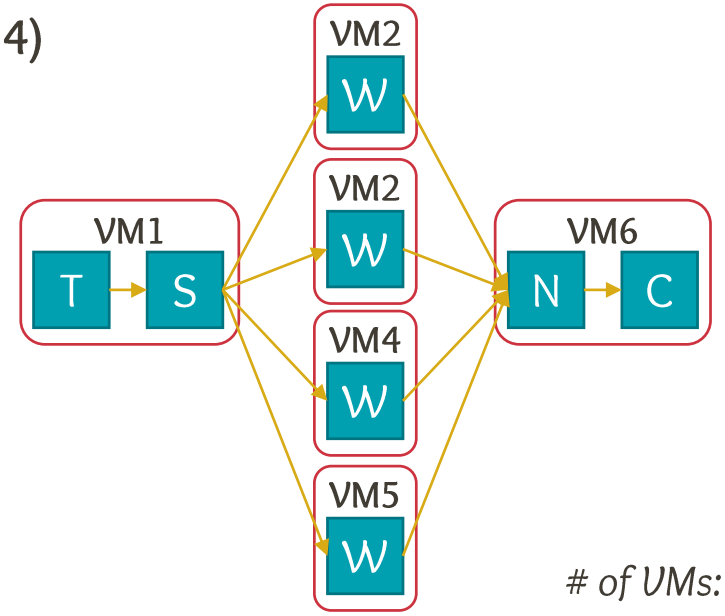


(3)



# of VMs: 2, 4, 6

(4)



# of VMs: 3, 4, 5, 6

What is the best way to design your application so that when you can strongly/weakly scale as VMs are added? Reduce latency? Improve utilization? Reduce co$t?

Applying Patterns to SiTA

- Consider times taken per task *(Queue, Compute, Disk, N/W, Overhead)*
- *Makespan* of the application
- Bytes of data exchanged at source & intermediate
- Analysis of times taken, bottlenecks
  - *"Measure twice, Cut once"*
- Analyse strong & weak scaling
  - With ↑ VMs & ↑ Data size
- *Does reality meet expectations?*

Ongoing Assignments

- Textbook reading, ch. 6
- Project outline due now.
- Mar 5: Mid-term (move from Mar 3?)
- Mar 10: Research Paper mid-term draft submission due.
- Mar 12/13: Project Mid-term Report & Demo