Application Emulators and Simulation Models

Tahsin M. Kurc

Computer Science Department,
University of Maryland,
College Park, MD

Our Target

- Large scale data-intensive parallel applications
  - Flexible, accurate, *scalable* models of applications
- Large scale machines
  - Fast and accurate simulation models
- Performance prediction in reasonable amount of time on workstations

Application Emulators

- Exhibits computational and access patterns that resemble patterns observed in the real application
- Provides a parameterized model of the application, for scaling and changing program details
- A simplified version (of both program and data)
- An executable (parallel or sequential) program

Why do we need application emulators?

- Problems with using traces from actual runs
  - Obtained for a single instance of application and machine configuration
  - Static, hard to represent dynamic nature of the program
  - May become very large
- Problems with running full application on simulator
  - May complicate the task of simulator unnecessarily
  - Execution of real application requires real data
  - Scaling real application for large scale machines may not be possible
- Application emulator
  - Parameterized, not specific to a single instance of application/machine configuration.
  - It is a *program*, so can model dynamic nature of application.
  - Level of abstraction can be controlled, simplifying task of simulator
  - Does not require real data, so can be scaled for large machines

Data-intensive Scientific Applications Suite

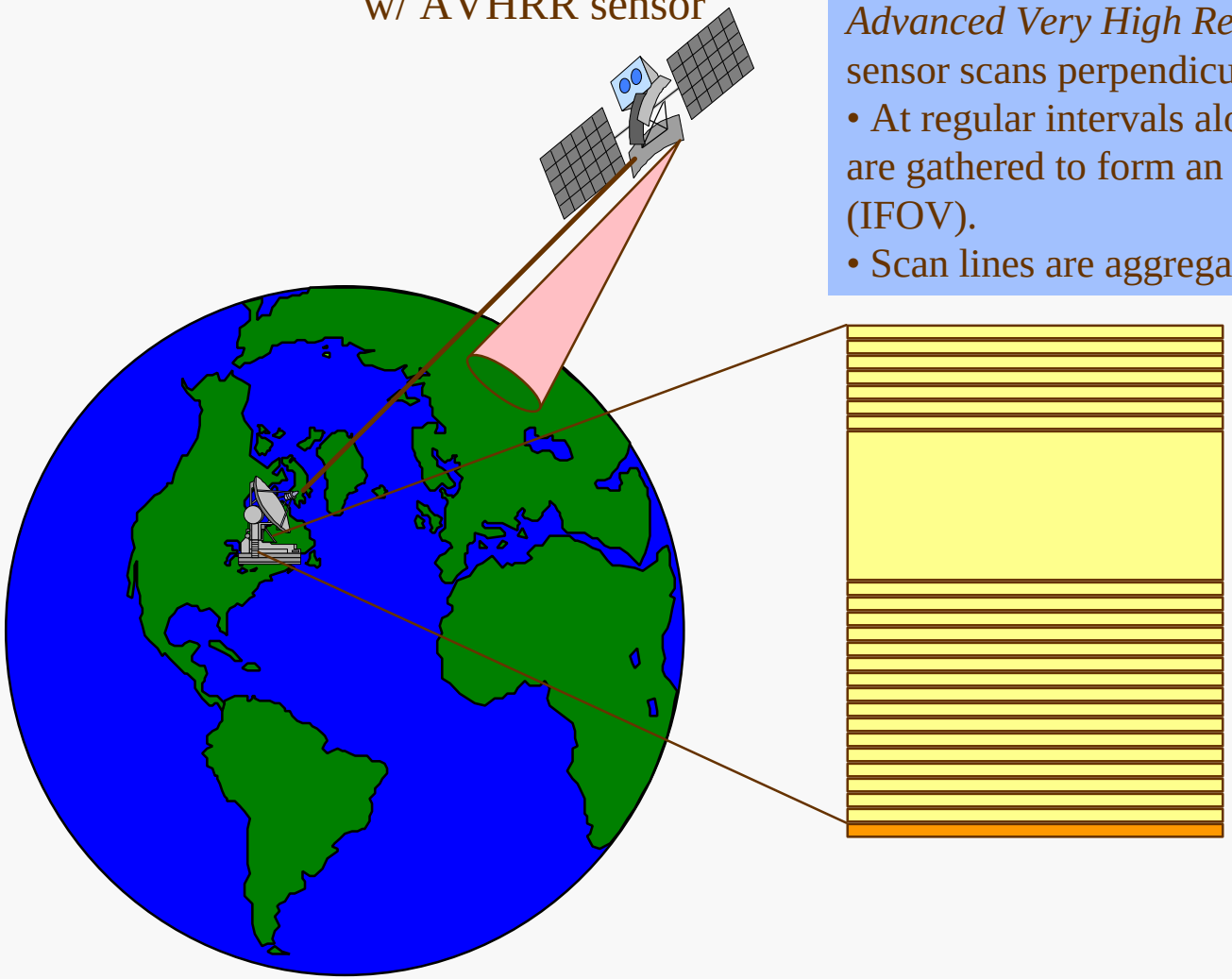
- Titan
  - Satellite data processing
  - peer-to-peer
- Pathfinder
  - Satellite data processing
  - client-server (separate I/O and Compute nodes)
- Virtual Microscope
  - Light microscope image database server
  - data server (multiple simultaneous queries), peer-to-peer

Titan: Input Data Structure

- Satellite Data
  - Satellite orbits earth in polar orbit
  - Each element (IFOV) is associated with a position (in longitude-latitude) and time of recording
- Input data is partitioned into data-blocks
  - Unit of I/O and communication is a data-block
  - Each block contains the same number of input elements
  - Spatial extent of each block varies
  - More overlapping blocks near the poles
- Data is distributed across disks for I/O parallelism
  - Minimax algorithm (Moon et al. 1996) for declustering

Remotely Sensed Data

NOAA Tiros-N
w/ AVHRR sensor



AVHRR Level 1 Data

- As the TIROS-N satellite orbits, the *Advanced Very High Resolution Radiometer* (AVHRR) sensor scans perpendicular to the satellite's track.
- At regular intervals along a scan line measurements are gathered to form an *instantaneous field of view* (IFOV).
- Scan lines are aggregated into Level 1 data sets.

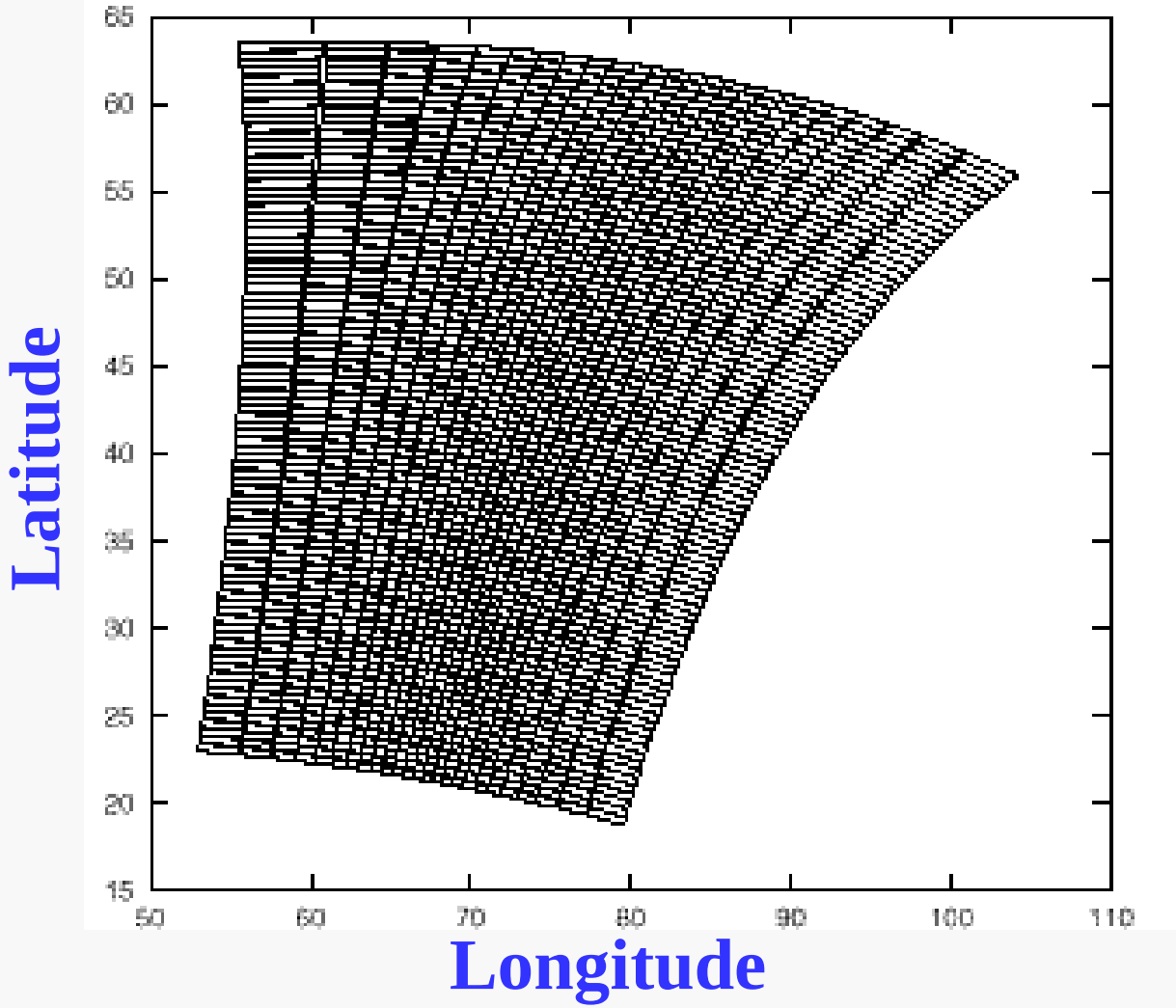
A single file of *Global Area Coverage* (GAC) data represents:

- ~one full earth orbit.
- ~110 minutes.
- ~40 megabytes.
- ~15,000 scan lines.

One scan line is 409 IFOV's

Spatial Irregularity

AVHRR Level 1B NOAA-7 Satellite 16x16 IFOV blocks.



Titan: Output Data Structure

- 2D image (latitude and longitude)
- Partitioned into equal size rectangles among processors
- Each processor is responsible for processing of blocks that map onto its region

Titan: Processing Loop

```
While (not done) do
    Issue reads
    Issue receives
    Poll reads
        if (some reads completed) then
            Map data-block to output data
            if (mapped to other processors)
                Issue sends to those processors
            if (mapped to myself)
                Enqueue for processing
    Poll receives
        if (data-block received)
            Enqueue for processing
    Poll sends
    Process a data-block
end while
```

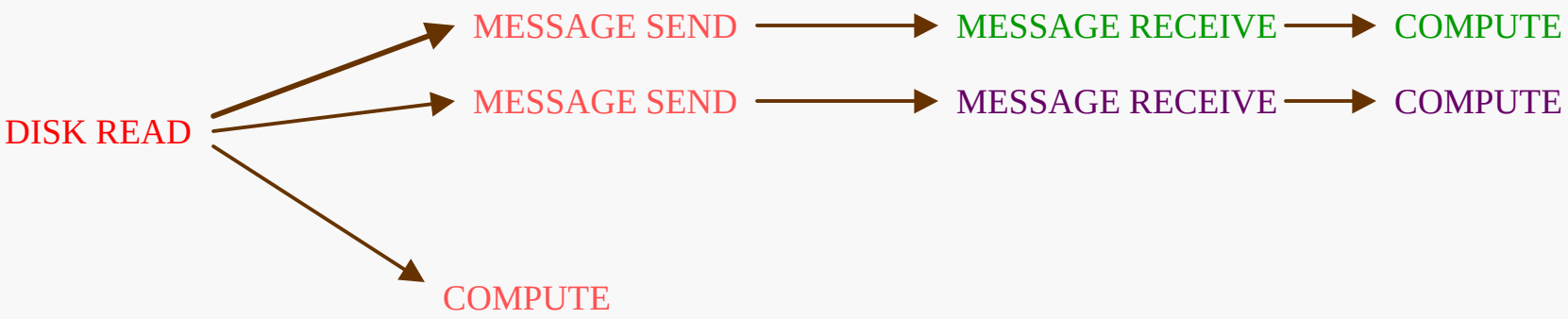
not done when there are

* reads yet to be issued
* pending reads
* receives yet to be issued
* pending receives
* pending sends
* blocks yet to be processed

Processing Loop

* All communication and I/O are non-blocking operations
* There are dependencies between operations on a data-block

Life cycle of a data-block



An Emulator for Titan

- Input Data Structure
  - I/O, Communication, Computation patterns
- Output Data Structure (Work load partitioning)
  - Communication, Computation patterns
- Processing Loop
  - I/O, Communication, Computation patterns

An Emulator for Titan

- Description of the machine
  - number of processors and disks
  - machine description file

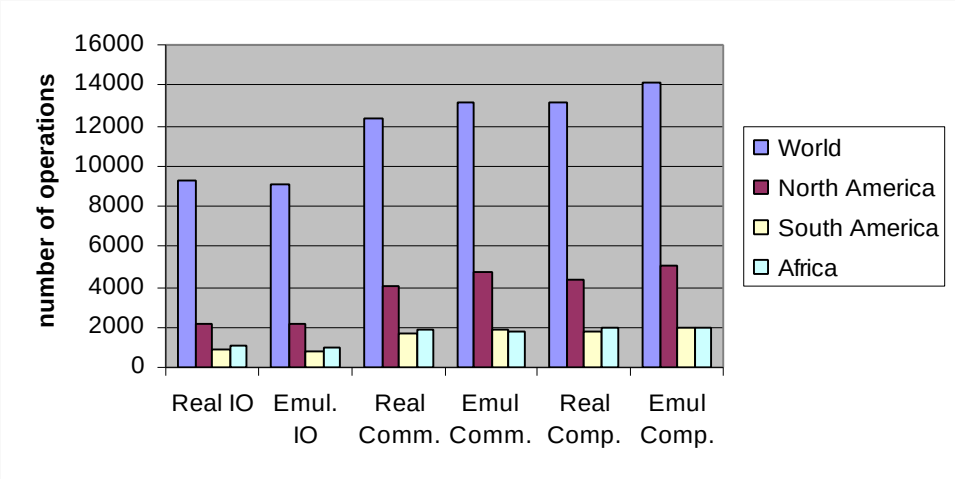
- Input Data Structure
  - A data-block is represented by a 2D rectangle
  - Controlled generation of data-blocks using functions
  - Parameterized generation of blocks
    - number of blocks
    - size of a block
  - Simple block-cyclic distribution of blocks to disks

An Emulator for Titan

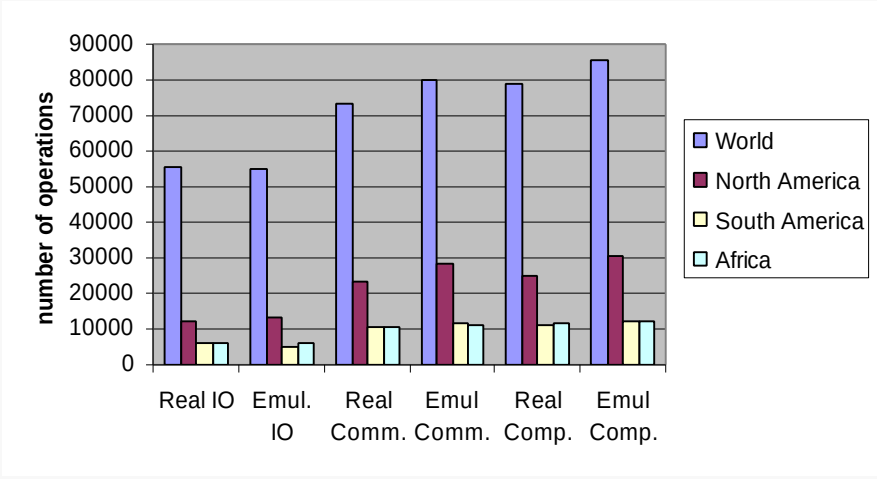
- Output Data Structure
  - Represented by a 2D rectangle
  - Parameterized 2D processor mesh
    - number of processors in x and y dimensions
- Processing Loop
  - Retain non-blocking nature of operations
  - Retain dependencies between operations on a block
  - Parameterization of some operations
    - number of maximum pending reads, receives
    - number of blocks processed per iteration of loop
  - Each block is assumed to take the same amount of time
    - computation time of a block can be changed

Comparison of Real Application and Emulator (Maryland 16 processor IBM SP2)

Number of operations, 10-day data

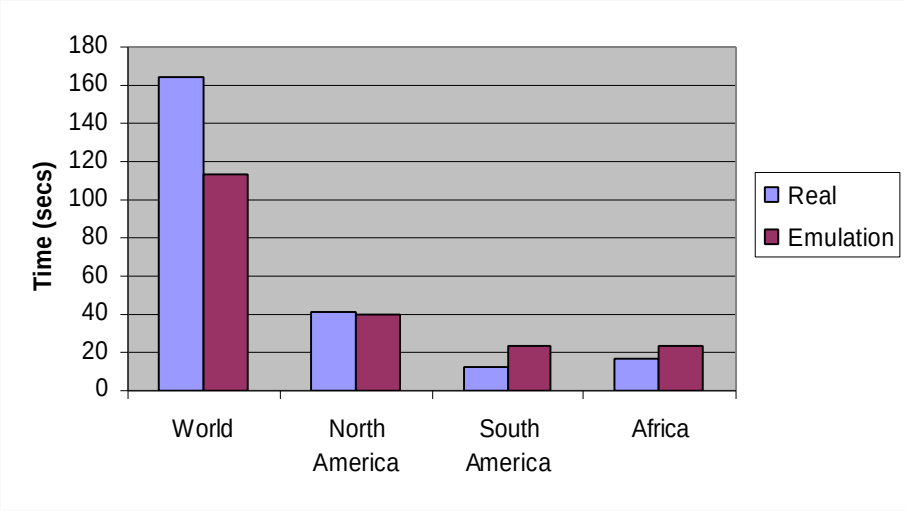


Number of operations, 60-day data

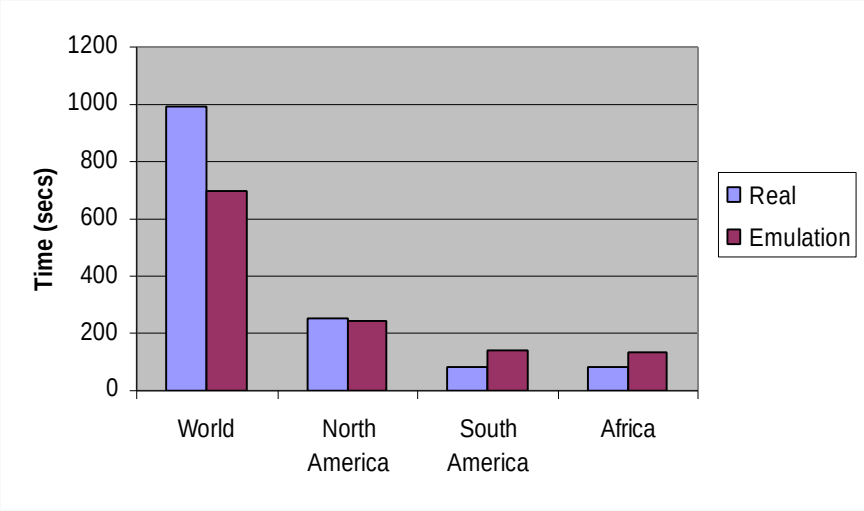


Comparison of Real Application and Emulator

Execution Times, 10-day data



Execution Times, 60-day data



Simulation Models
(Interaction between Emulators and Simulators)

- Tightly-coupled Simulation
  - Similar to running on real machine
    - a thread is created for each application emulator process
    - emulator performs calls to simulator API for
      - initiating I/O, communication, and computation operations (events)
      - checking their completion
  - Simulator schedules emulator threads to ensure correct logical order of operations
  - Emulator and simulator interacts for each event (e.g., disk read request)
  - Emulator keeps track of dependencies between operations

Simulation Models

Tightly-coupled simulation is not suitable for simulating large scale machines

- Number of emulator threads increases with increasing number of processors
  - Scheduling these threads becomes very costly
- Message and I/O tables for outstanding non-blocking operations become very large
  - Need for large memory to store these tables
  - Very costly to manage these tables
- Each emulator thread has to keep track of non-blocking operations
  - Needs its local data structures (tables) for these operations
  - Replicates the work of simulator.

Tightly-coupled Simulation

- Predicted execution time and simulation time for tightly-coupled simulation
- Measurements done on Digital Alpha 2100 4/275 workstations
- All results are in seconds
- Machine parameters for simulator are the same as Maryland IBM SP2

Emulator	Dataset	P	Predicted Application Execution Time	Execution Time of Simulator
	27K blocks	32	211	3426
Titan	55K blocks	64	285	13154
	110K blocks	128	604	**116224**
	55K blocks	32	551	11595
Pathfinder	110K blocks	64	718	30446
	220K blocks	128	1020	**97992**
	500 K blocks	32	135	7155
Virtual Microscope	1000K blocks	64	145	14097
	2000K blocks	128	158	**37534**

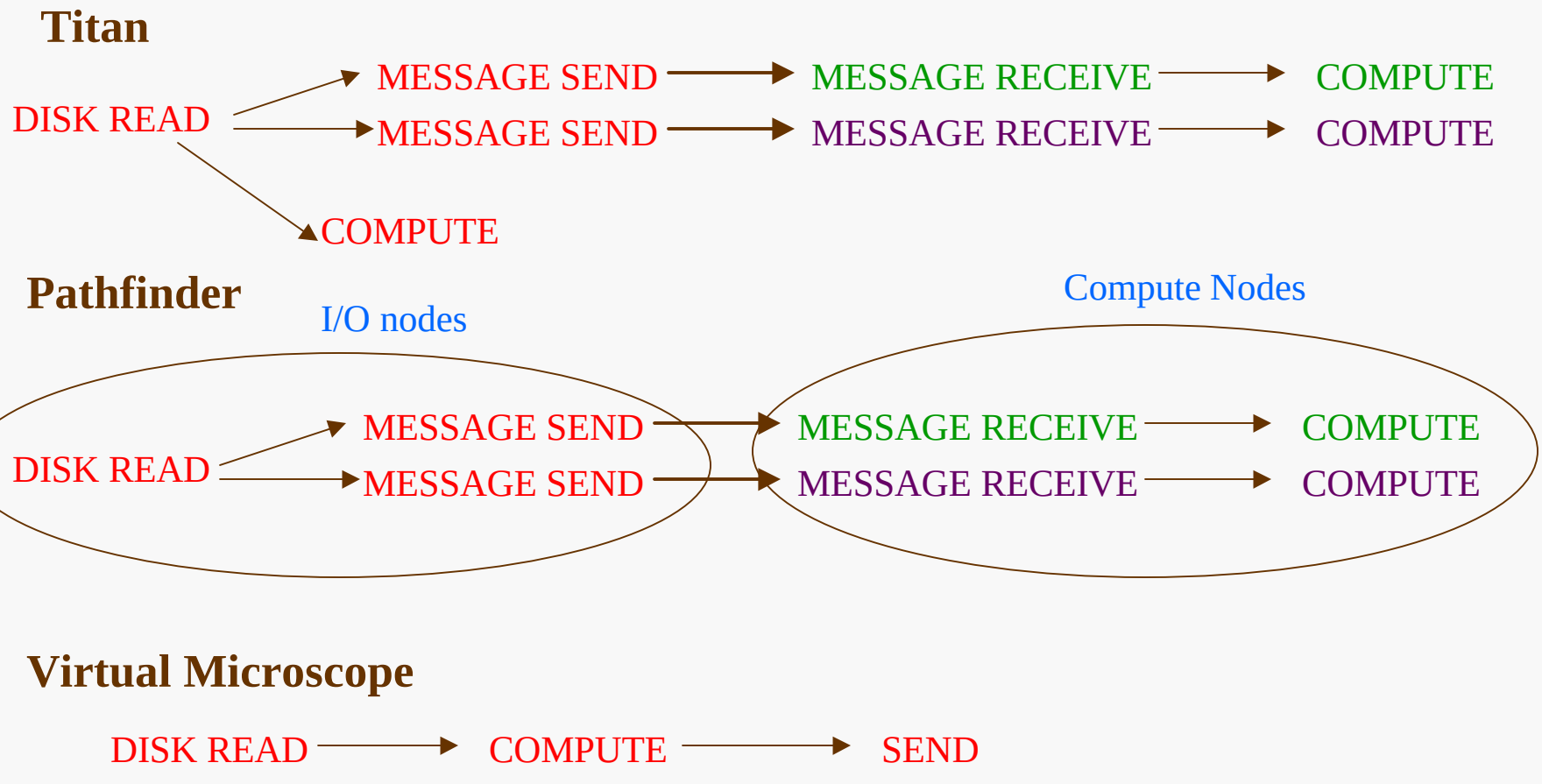
Simulation Models

- Loosely-coupled Simulation
  - Idea: Embed application processing loop into simulator
    - Dependency information of processing loop is embedded in the simulator
  - A Model of the application processing structure
    - Work flow graphs
  - Two separate processes - one for emulator, one for simulator
  - Emulator and simulator interact in distinct phases
    - Epoch-based interaction

Simulation Models
(Modeling Dependencies: Work Flow Graphs)

- Describes the dependencies between operations performed on a data-block
  - Nodes are the operations performed on the data-block
  - Directed edges are the dependencies between these operations
  - Imposes a partial order of events.
- Basic skeleton of the work flow graph depends on the characteristics of the application
- Parameterized by the emulator to reflect specific instance of
  - input data
  - output data
  - machine configuration

Simulation Models
(Work Flow Graphs for Current Applications Suite)



Simulation Models
(Epoch-based interaction)

- The emulator and the simulator interact in distinct phases
  - Emulator sends a set of events (for a set of blocks) to the simulator
  - Simulator processes these events
  - Simulator asks for another set of events from emulator
- For each block in a set for each processor, emulator passes to simulator
  - disk id
    - indicates a read operation from that disk
  - length of the block
    - used to estimate I/O and communication time
  - list of consumers
    - indicates communication (sends and receives)
  - computation time of the block

Conclusions and Future Work

- Emulators for Data-intensive scientific applications
  - Simple and parameterized model of applications
  - Enables performance prediction studies for large scale applications on large scale machines
- Loosely-coupled simulation
  - Enables simulation of large scale machines
- We were able to
  - model application datasets up to 384 terabytes in size
  - run simulations for machines up to 8K processors and 32K disks on workstations

Conclusions and Future Work

- Ongoing and Future Work
  - Different simulators
    - Howsim, Gigasim (Mustafa Uysal, University of Maryland)
    - Dumbsim, Fastsim (Jeff Hollingsworth and Hyeonsang Eom, University of Maryland)
    - Petasim (Geoffrey Fox and Yuhong Wen, Syracuse University)
  - Generalizing Work Flow Graphs
    - Application emulators for different classes of applications
  - Automated generation of application emulators/work flow graphs (Apostolos Gerasoulis, Rutgers University)