# Module E:
# Distributed Scientific Computing

Introduction to M-W Pattern,
MapReduce and Cloud Computing

Dr Shantenu Jha

http://radical.rutgers.edu

# Overview of Module E
# Distributed Scientific Computing

- Introduction to M-W and Cloud Computing
  - Master-Worker Pattern
  - Examples of M-W Pattern:
    - M-W Example Using SAGA: Mandelbrot Set
    - Ensemble simulations, Replica-Exchange
    - Introduction to MapReduce
    - Wordcount using SAGA MapReduce
- Introduction to Cloud Computing
    - Why Cloud Computing?
      - Convergence of multiple trends: Data-centric, Data-Center…
    - Understanding Amazon EC2 – default 'standard'

# Master-Worker Pattern

- Pattern: A commonly occurring mode of computation
  - Multiple patterns exist
    - e.g., publish-subscribe, broker etc.,
    - But M-W arguably one of the most pervasive
- M-W: Used in parallel and distributed computing
  - Simply put: Master assigns task to a worker; worker does work; gets more from Master
    - Master coordinates task distribution
  - M-W not an application in of itself, but a programming model or "communication pattern" upon which applications can be built
- What types of tasks are suitable for M-W?
  - Many independent "units" of loosely coupled tasks
  - Concurrent execution is feasible/permissible

# Master-Worker Pattern

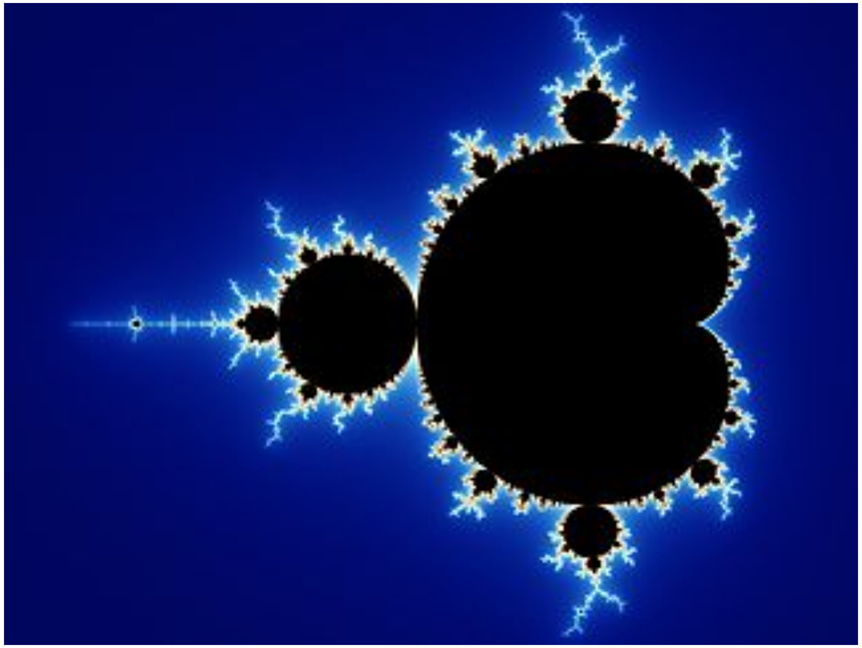
- What types of tasks are not suitable for M-W?
  - Decomposing into smaller independent units is not trivial
  - Lots of communication:
    - Either between Master and a Worker (s)
      - Master becomes the bottleneck!
    - Or between workers?
- Of Applications in E1, which are/can be M-W?
  - Nektar? Montage? SCOOP? Climateprediction.net?
  - Ensemble simulations and/or Replica-Exchange

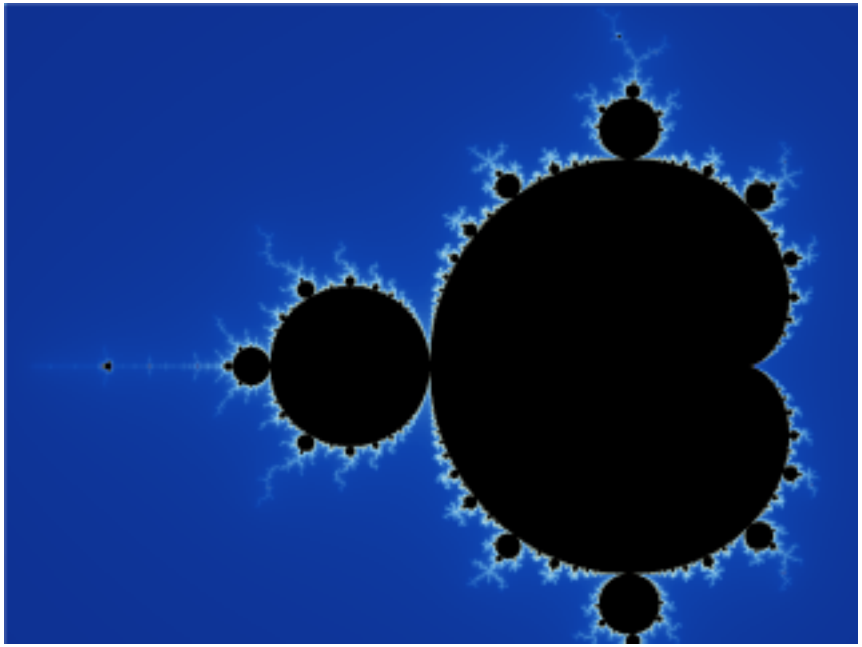
# Some Challenges in Distributed M-W Execution

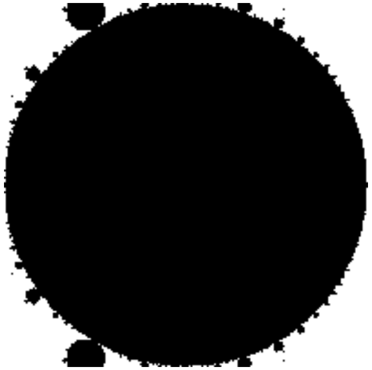
- Task Decomposition and coordination:
  - How do we assign work units to workers?
  - What if we have more work units than workers?
- Execution and Fault-Tolerance:
  - How do we know all the workers have finished?
  - What if workers die?
- Coordination:
  - What if workers need to share partial results?
  - How do we aggregate partial results?
- Q: Based upon the above, what other constraints on suitability for M-W?

# MANDELBROT SET

# M-W to Compute Mandelbrot Set







How is M-W used to compute Mandelbrot Set?

- Task item: Complex plane broken-up; compute parts of it
- Master puts task items into bucket. Worker collects tasks;

# SAGA-Based M-W: Mandelbrot

- You've seen Mandelbrot using SAGA-Python and BigJob
- Q: Discuss the similarities and the differences? Are they both implemented as a M-W pattern?

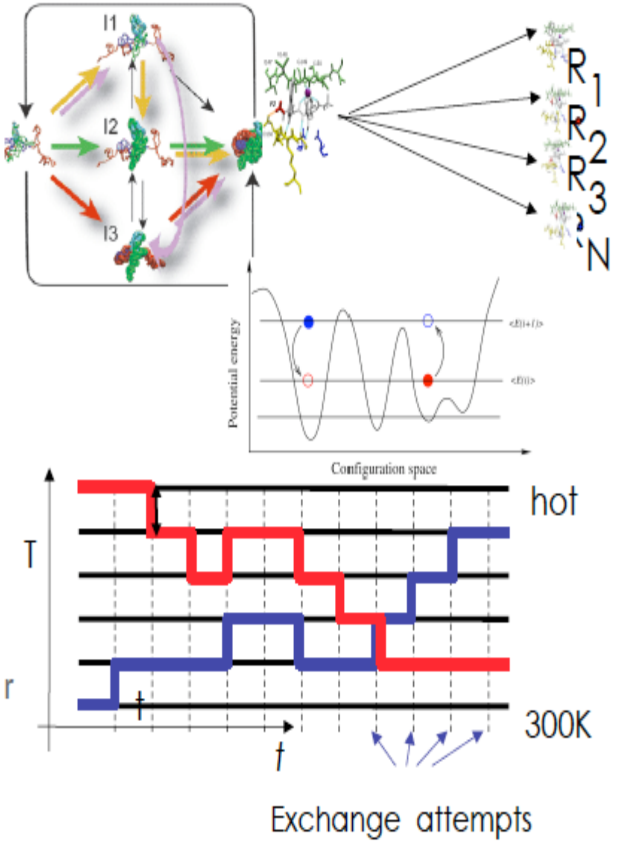
# SAGA-Based M-W: Mandelbrot

1. Everything local: For 1 Master and same workload vary: $N_w$ = 2, 4 and 8  Plot times to completion.
2. Distribute (equally?) the workers across a couple of XSEDE machines. Compare with (i) and (2)

# ENSEMBLE-BASED REPLICA-EXCHANGE

# Ensemble-based & Replica-Exchange Simulations

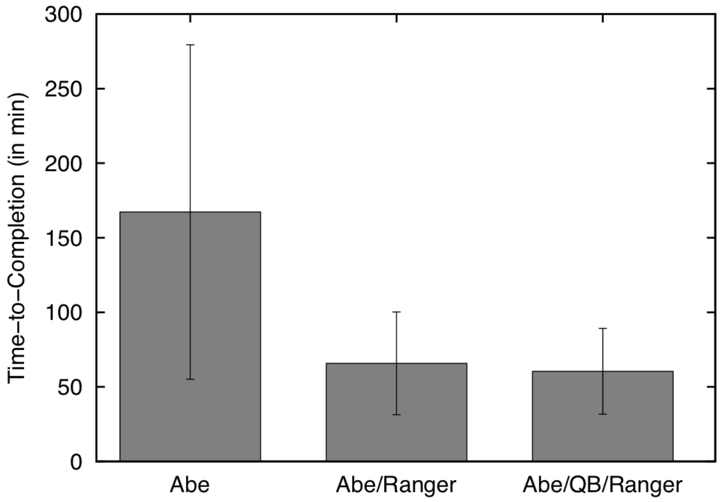
- Ensemble-based:
  - Many uncoupled simulations
  - But not necessarily uncoupled in analysis!
- Replica-Exchange (RE) methods:
  - Represent a class of algorithms that involve a large number of loosely coupled ensembles.
- RE simulations are used to understand a range of physical phenomena
  - Protein folding, unfolding etc
  - MC simulations
- Many successful implementations
  - Eg folding@home [replica based]

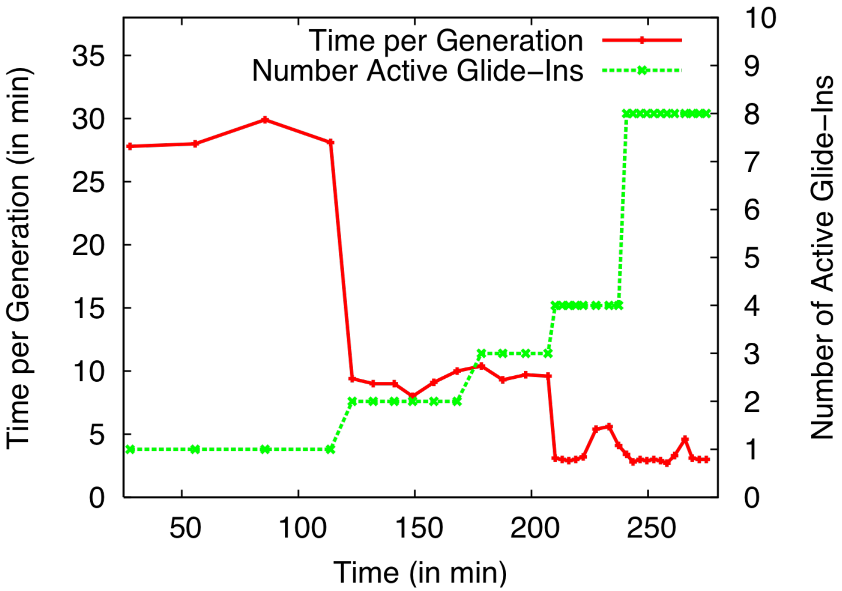


# Distributed Adaptive Replica Exchange (DARE)

## Multiple Pilot-Jobs on the "Distributed" TeraGrid

- Ability to dynamically add HPC resources. On TG:
  - Each Pilot-Job 64px
  - Each NAMD 16px
- Time-to-completion improves
  - No loss of efficiency

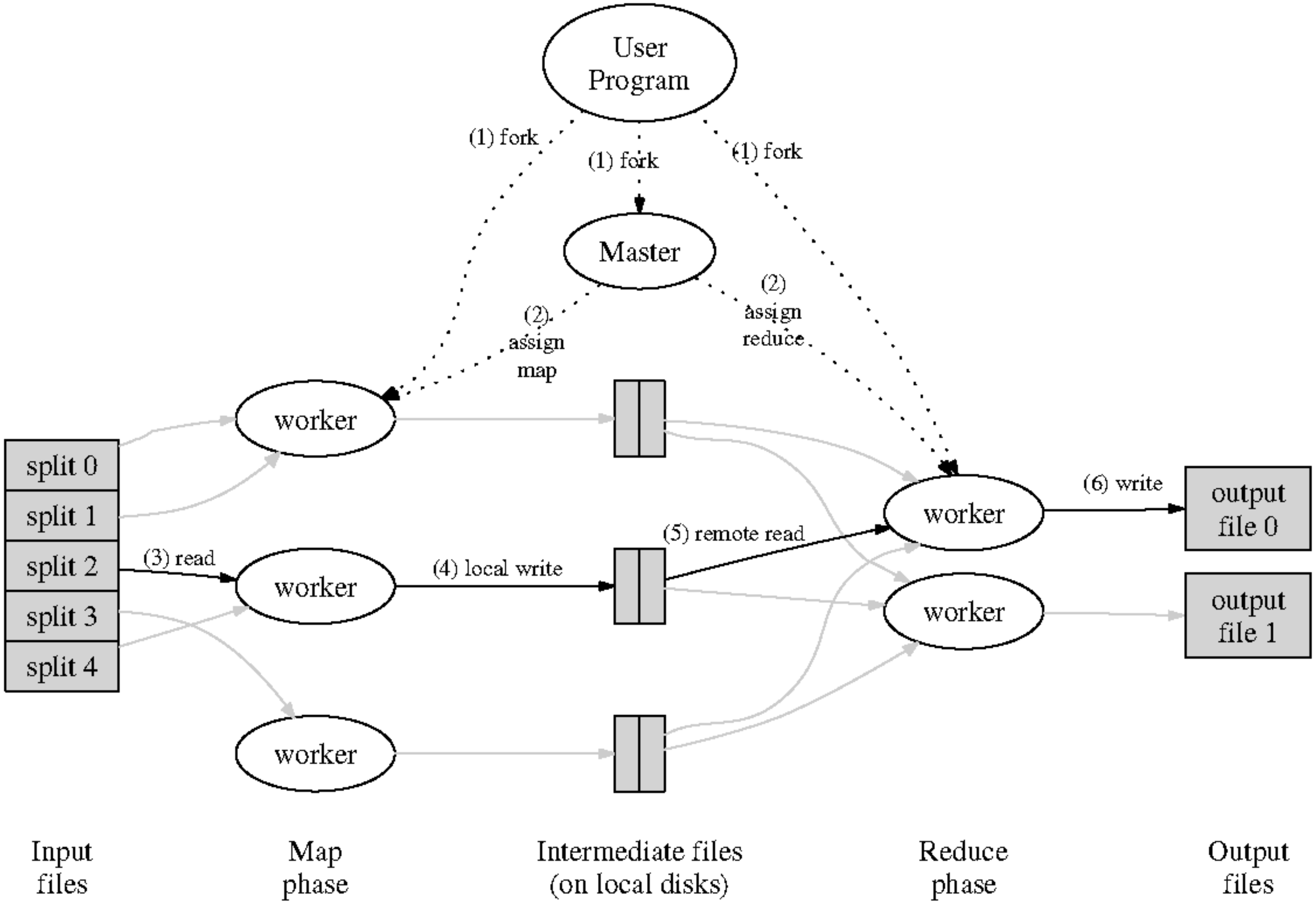




# Understanding Replica-Exchange

- Why Distributed?
  - Many un-coupled units (ensembles/replica)
  - More resources, the merrier!!
- How Distributed?
  - Many implementations exist (eg folding@home)
  - SAGA-based "Pilot-Jobs" to use many distributed TG resources
- Limitations and Success?
  - Getting SAGA working on all machines!
  - Finding the best set of resources
  - Coordinating work across all the resources

# MAP-REDUCE



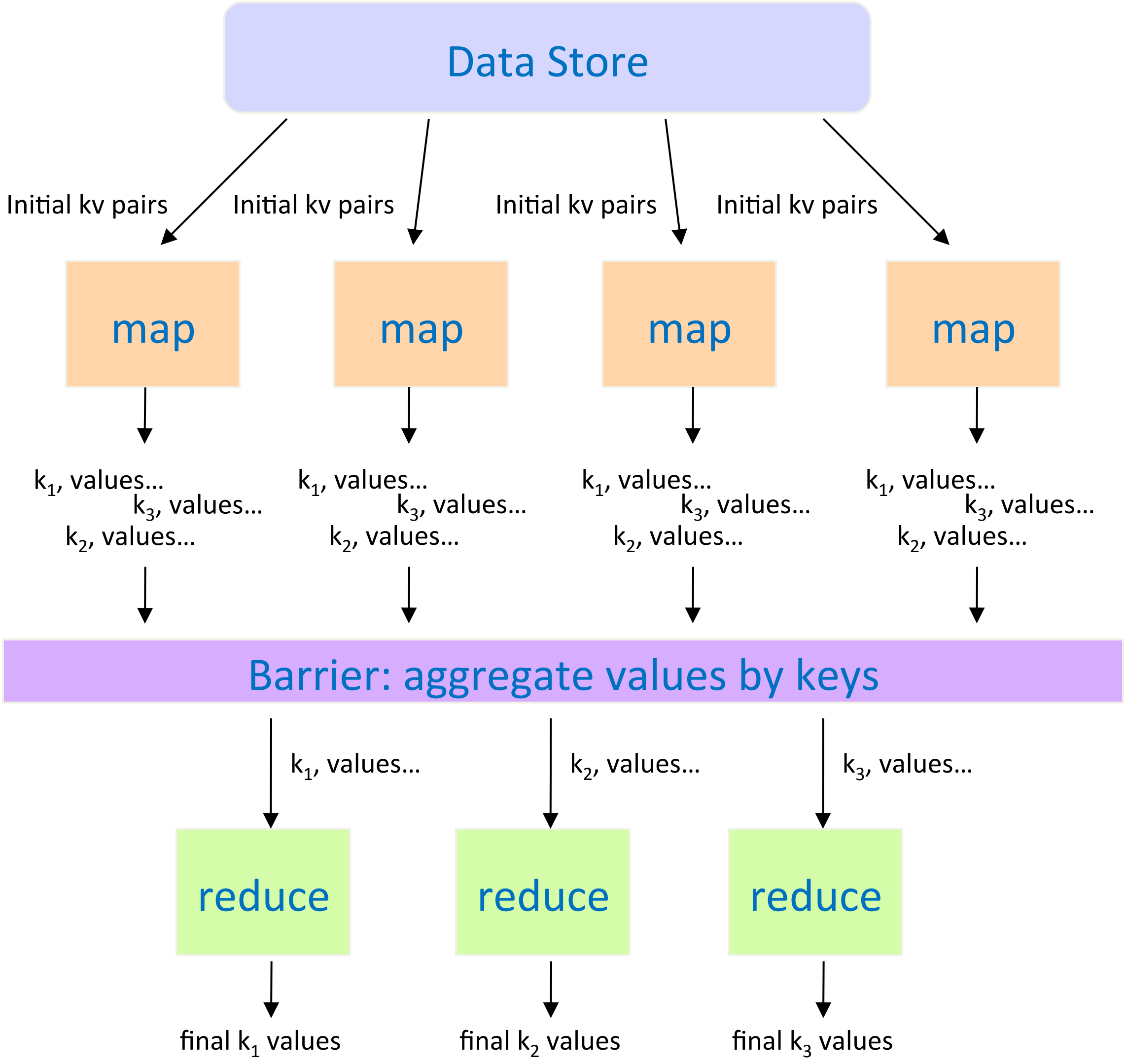
Source: Dean and Ghemawat (OSDI 2004)

# “Hello World”: Word Count

```
Map(String input_key, String input_value):
    // input_key: document name
    // input_value: document contents
    for each word w in input_values:
        EmitIntermediate(w, "1");

Reduce(String key, Iterator intermediate_values):
    // key: a word, same for input and output
    // intermediate_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
        result += ParseInt(v);
        Emit(AsString(result));
```

# Word Count via MapReduce



Data Store

Initial kv pairs

Initial kv pairs

Initial kv pairs

Initial kv pairs

map

map

map

map

$k_1$, values…
$k_3$, values…
$k_2$, values…

$k_1$, values…
$k_3$, values…
$k_2$, values…

$k_1$, values…
$k_3$, values…
$k_2$, values…

$k_1$, values…
$k_3$, values…
$k_2$, values…

Barrier: aggregate values by keys

$k_1$, values…

$k_2$, values…

$k_3$, values…

reduce

reduce

reduce

final $k_1$ values

final $k_2$ values

final $k_3$ values

# Some Challenges in Distributed M-W Execution (Redux)

- Task Decomposition and coordination:
  - How do we assign work units to workers?
  - What if we have more work units than workers?
- Execution and Fault-Tolerance:
  - How do we know all the workers have finished?
  - What if workers die?
- What if workers need to share partial results?
- How do we aggregate partial results?

# MapReduce versus Google MapReduce (Runtime)

- MapReduce the pattern versus MapReduce the runtime
- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles "data distribution"
  - Moves the process to the data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed FS (later)

# WORDCOUNT USING SAGA MAPREDUCE

# SAGA MAPREDUCE

- Not tied to any specific infrastructure
  - Interoperable across different back-ends
  - No optimization, thus performance barrier
- Can control chunk-size, task size granularity, decomposition and placement/distribution
- Master-Worker pattern
  - Uses Advert Service to coordinate and distribute
- Contrast with Google MapReduce or Hadoop
  - Google/Yahoo extensively use the File-System
  - SAGA's flexibility comes at a performance!

# SAGA PMR

- SAGA-based (Pilot) MapReduce:
  - https://github.com/saga-project/PilotMapReduce

# SAGA PMR: WORDCOUNT

- Word Count:
  - https://github.com/saga-project/PilotMapReduce/tree/master/applications/wordcount
- *Generate your own input file for the wordcount example*
  1. Everything local: For 1 Master and same workload vary: $N_w$ = 2, 4 and 8  Plot times to completion.
  2. Distribute (equally?) the workers across a couple of XSEDE machines. Compare with (i) and (2)
  3. Describe the role of the Pilot-Job?

# Cloud Computing

# What is cloud computing?

- *I don't understand what we would do differently in the light of Cloud Computing other than change the wordings of some of our ads*

Larry Ellision, Oracle's CEO

- *I have not heard two people say the same thing about it [cloud]. There are multiple definitions out there of "the cloud"*

Andy Isherwood, HP's Vice President of European Software Sales

- *It's stupidity. It's worse than stupidity: it's a marketing hype campaign.*

Richard Stallman, Free Software Foundation founder

# What is a Cloud?

## From NIST

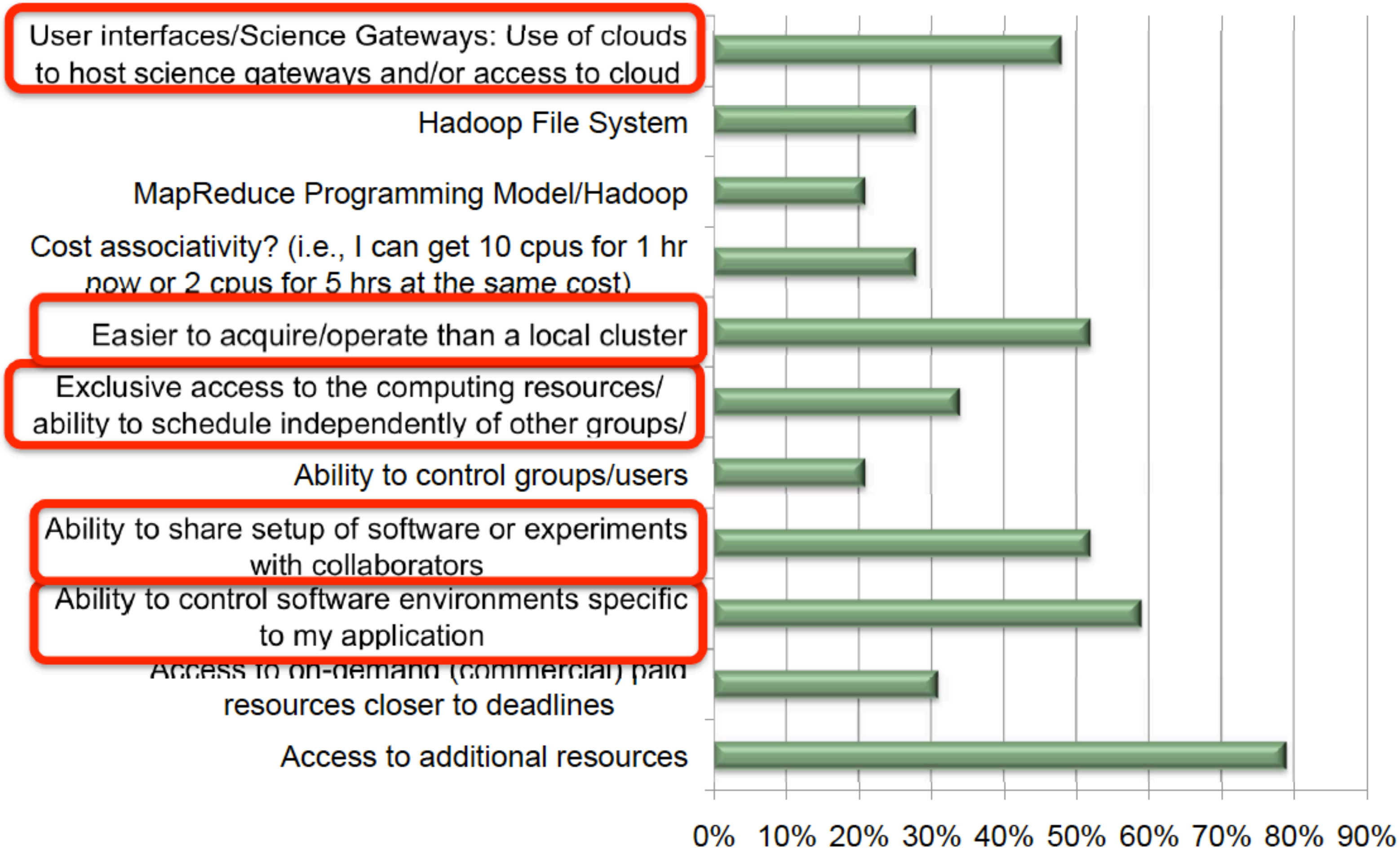
- *Resource pooling.* **Computing resources are pooled to serve multiple consumers.**
- *Broad network access*. **Capabilities are available over the network.**
- *Measured Service.* **Resource usage is monitored and reported for transparency.**
- *Rapid elasticity.* **Capabilities can be rapidly scaled out and in (pay-as-you-go)**
- *On-demand self-service.* **Consumers can provision capabilities automatically.**

# Why is This not Good Ol' Supercomputing

- A Supercomputer is designed to scale a single application for a single user.
  - Optimized for peak performance of hardware.
  - Batch operation is not "on-demand".
  - Reliability is secondary
    - If MPI fails, app crashes. Build checkpointing into app.
  - Most data center apps run continuously (as services)
- Yet, "in many ways, supercomputers and data centers are like twins separated at birth."*

* Dan Reed

# Cloud Computing Interest
(adapted from Kathy Yelick)



# Cloud Models

**Infrastructure as a Service**

- Provide a way to host virtual machines on demand
  - Amazon ec2 and S3 – you configure your VM, load and go

**Platform as a Service**

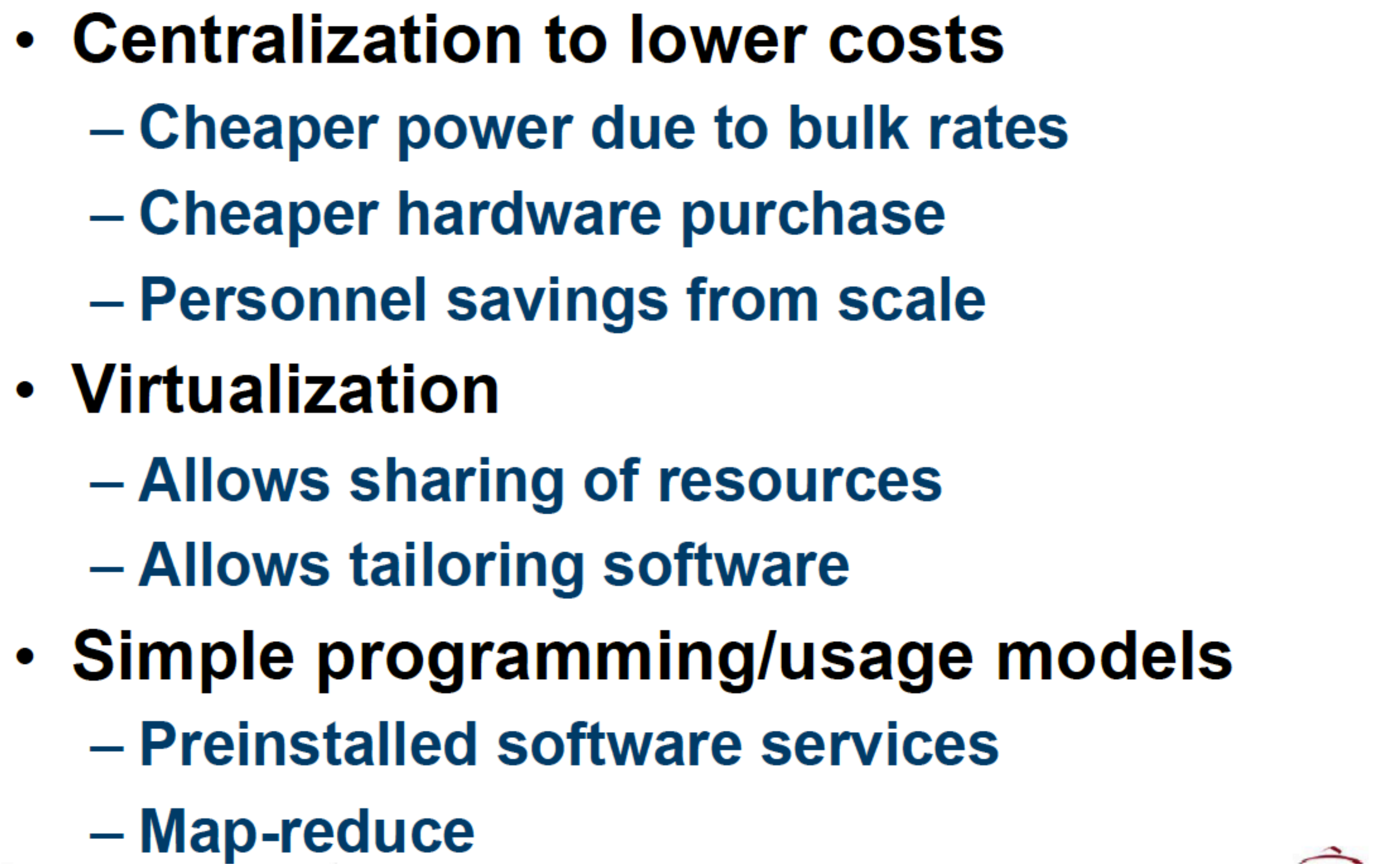
- You write an App to cloud APIs and release it. The platform manages and scales it for you.
- Google App engine:
  - Write a python program to access Big Table. Upload it and run it in a python cloud.
  - Hadoop and Dryad are application frameworks for data parallel analysis

**Software as a Service**

- Delivery of software to the desktop from the cloud
  - Stand-alone applications (Word, Excel, etc)
  - Cloud hosted capability
    - doc lives in the cloud
    - Collaborative document creation

- For more details on *aaS see paper by Lamia Youssef and Rich Wolski (GCE'09 @ SC09)

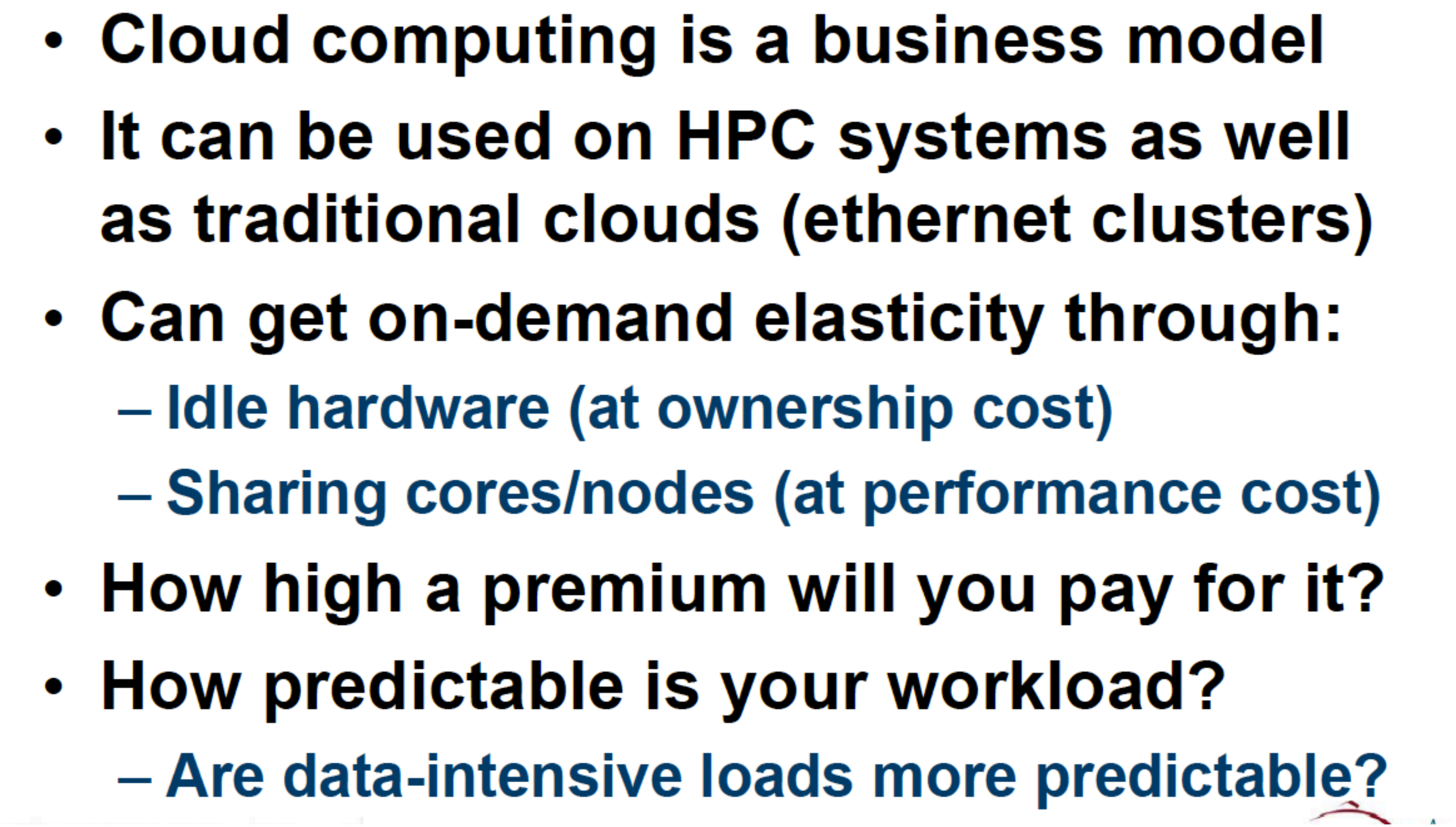
# Cloud Computing: Enabling Technologies
(adapted from Kathy Yelick)



- **Centralization to lower costs**
  - Cheaper power due to bulk rates
  - Cheaper hardware purchase
  - Personnel savings from scale
- **Virtualization**
  - Allows sharing of resources
  - Allows tailoring software
- **Simple programming/usage models**
  - Preinstalled software services
  - Map-reduce

# .. Its all about the Buisiness Model

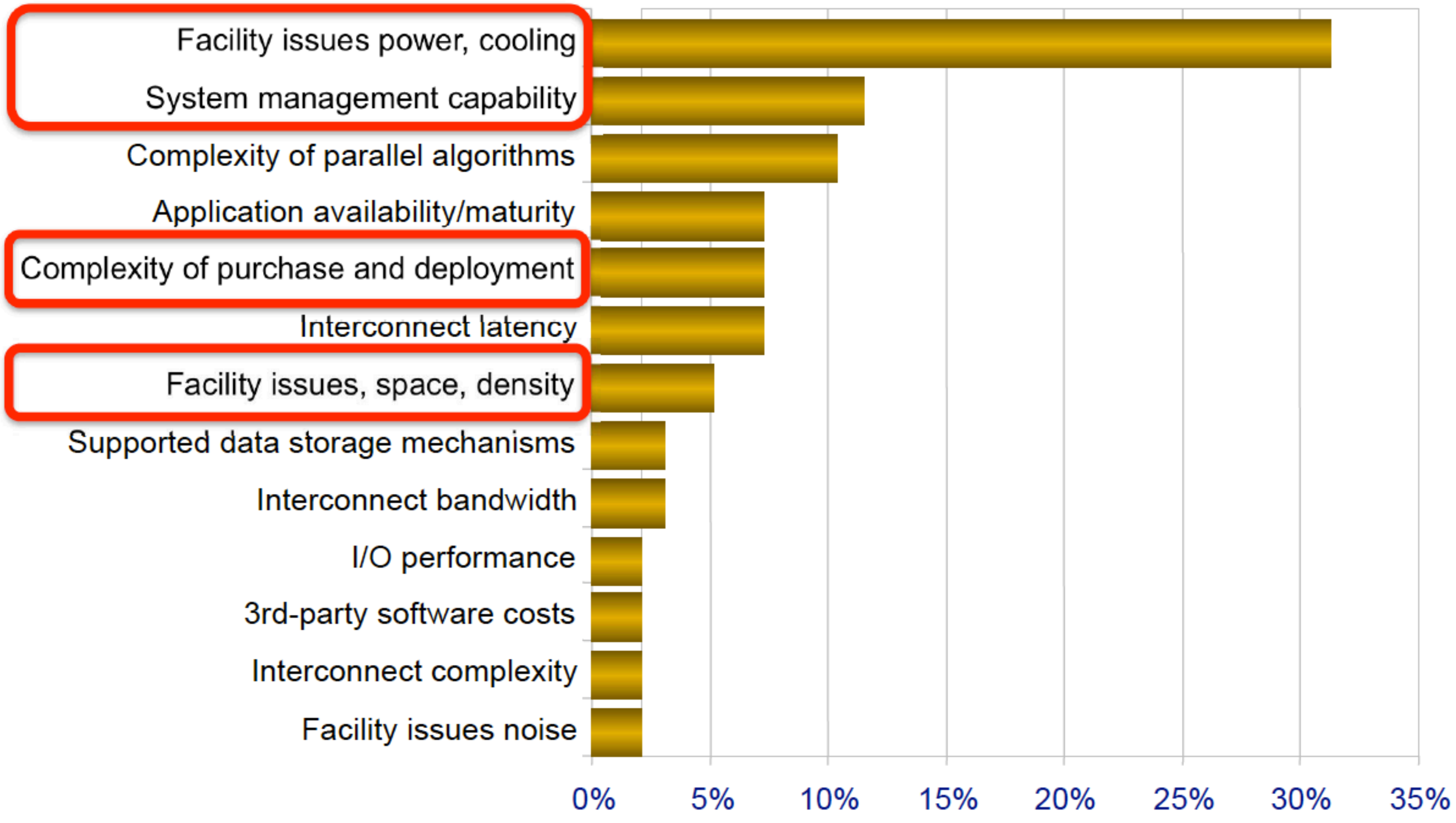
(adapted from Kathy Yelick)



- Cloud computing is a business model
- It can be used on HPC systems as well as traditional clouds (ethernet clusters)
- Can get on-demand elasticity through:
  - Idle hardware (at ownership cost)
  - Sharing cores/nodes (at performance cost)
- How high a premium will you pay for it?
- How predictable is your workload?
  - Are data-intensive loads more predictable?

# Top challenges to running own cluster

(adapted from Kathy Yelick)



n = 96

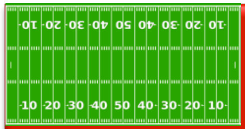
# The Data Center Landscape

Range in size from "edge" facilities to megascale.

Economies of scale

Approximate costs for a small size center (1K servers) and a larger, 50K server center.

| Technology | Cost in small-sized Data Center | Cost in Large Data Center | Ratio |
|---|---|---|---|
| Network | $95 per Mbps/ month | $13 per Mbps/ month | 7.1 |
| Storage | $2.20 per GB/ month | $0.40 per GB/ month | 5.7 |
| Administration | ~140 servers/ Administrator | >1000 Servers/ Administrator | 7.1 |



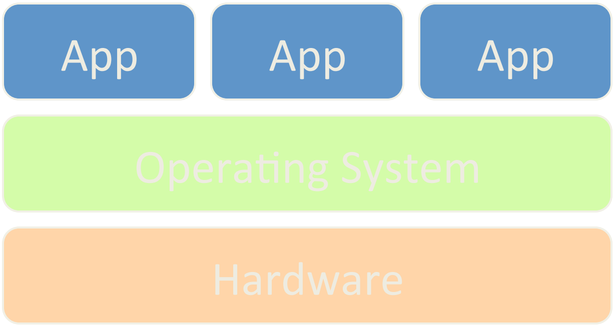
Each data center is **11.5 times** the size of a football field

# Advances in DC deployment

Conquering complexity.

- Building racks of servers and complex cooling systems all separately is not efficient.
- Package and deploy into bigger units:

# Key Technology: Virtualization

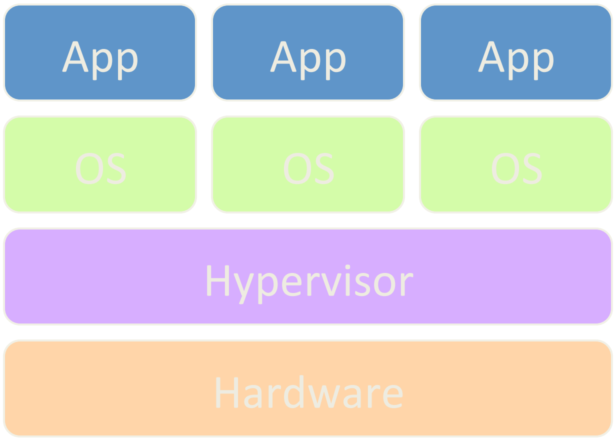


App App App

Operating System

Hardware

Traditional Stack



App App App

OS OS OS

Hypervisor

Hardware

Virtualized Stack

# Amazon AWS
## http://aws.amazon.com

- Story goes: Build capacity for X-mas. What do with spare capacity year around?
- "Utility Computing"
  - Around long before Amazon EC2
  - $0.10 per CPU-hour, plus bandwidth cost
- *aaS Model:
  - * = Infrastructure, Software, almost anything
- AWS: A set of APIs which give users access to Amazon technology and content
  - IaaS, but also "people as a service" – Mechanical Turk

# Amazon Simple Storage Service (S3)

- Data Storage in Amazon Data Center
- Web Service interface
- No set-up fee, No monthly minimum
- Storage: $0.15 per GB/Month
- Data Transfer: $0.20/GB to transfer data
- Private and  public storage
- Each object up to 5GB in size

# Amazon Elastic Compute Cloud

- A Web service that provides resizable compute capacity in the cloud. Designed to make Web-scale computing easier
- A simple Web service interface that provides complete control of your computing resources
- Quickly scales capacity, both up and down, as your computing requirements change
- Changes the economics of computing:
  - Pay only for capacity that used; no cost of ownership
    - *a + bc* becomes just *bc*

# Amazon Elastic Compute Cloud

- No start-up, monthly, or fixed costs
  - $0.10 per CPU hour
  - $0.20 per GB transferred across Net
- No cost to transfer data between Amazon S3 and Amazon EC2
- More when we do Cloud Computing next week

# Amazon Web Services

## Default “community” standard

- Compute
  - Elastic Compute Service (EC2)
  - Elastic MapReduce
  - Auto Scaling
- Storage
  - Simple Storage Service (S3)
  - Elastic Block Store (EBS)
  - AWS Import/Export
- Messaging
  - Simple Queue Service (SQS)
  - Simple Notification Service (SNS)
- Database
  - SimpleDB
  - Relational Database Service (RDS)
- Content Delivery
  - CloudFront
- Networking
  - Elastic Load Balancing
  - Virtual Private Cloud
- Monitoring
  - CloudWatch

http://aws.amazon.com/

# Elastic Compute Cloud (EC2) Service

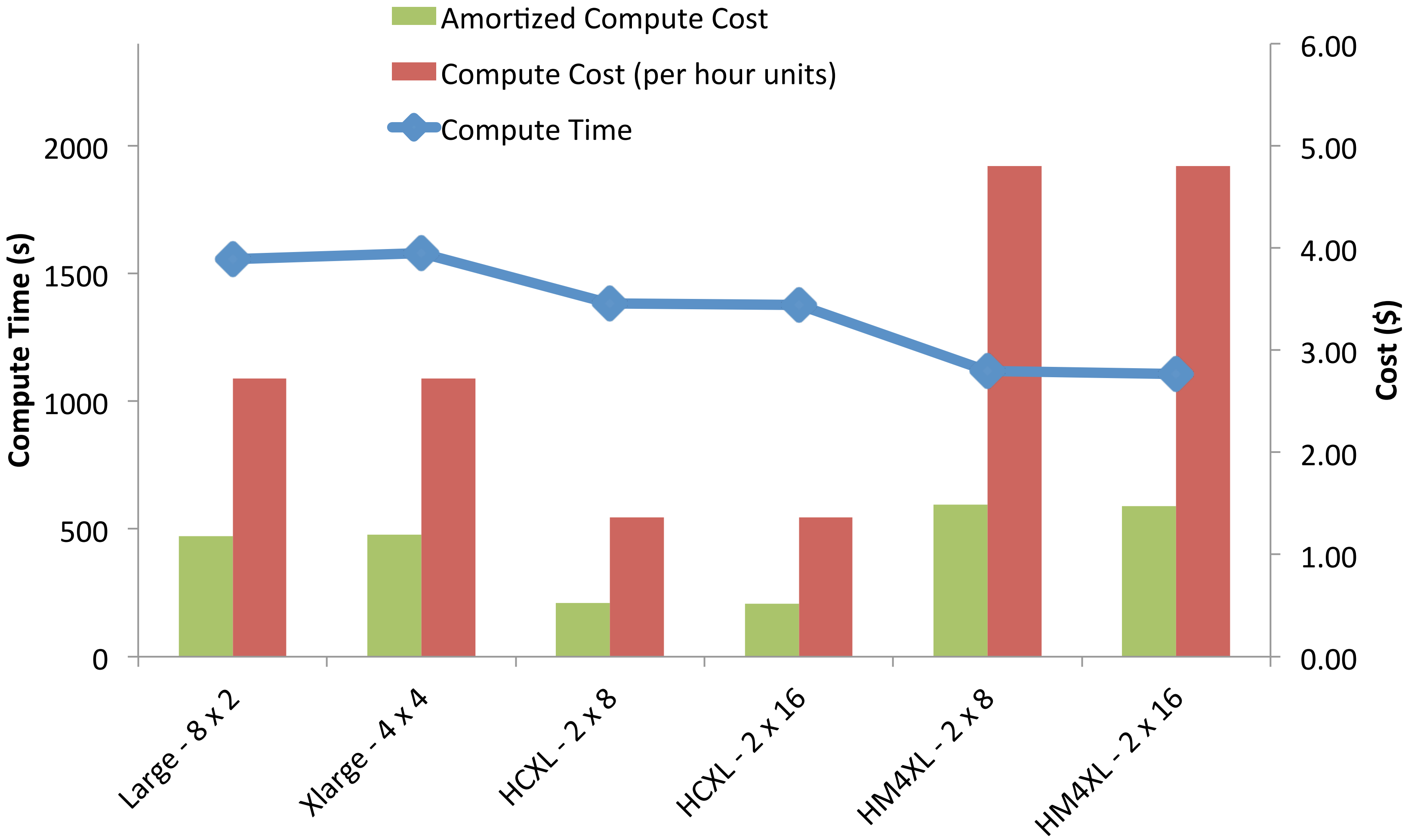
- Amazon Machine Image (AMI) is a special type of pre-configured operating system and virtual application software which is used to create a VM within EC2
  - Use either Pre-configured, templated images or create AMI to store customized images. Can share AMI (via AMI ID)
  - It serves as the basic unit of deployment for services delivered using EC2. Lease Linux as well as Windows AMI
  - See http://aws.amazon.com/amazon-linux-ami/
- VM = Bind an AMI to an Instance
  - Multiple Instance Types (see next slide)
  - Dynamically scale up/down
  - ‘root’ access to VM’s

# Elastic Compute Cloud (EC2) Service

- EC2 Instances Types
    - http://aws.amazon.com/ec2/instance-types/
    - Standard Instance
        - Small, Large and Extra-Large
    - Micro Instance
    - High-Memory Instances
        - XL, Double XL, Quadruple XL
    - High-CPU Instances
        - High-CPU Medium, High-CPU XL
    - Cluster Compute Instance
        - Cluster Compute Quadruple
    - Cluster GPU  Instance
    - …

# Sequence Assembly Performance with different EC2 Instance Types

## (Adatped From Geoffrey Fox)



# Azure

- Description: Microsoft's "Platform as a Service" (Paas) offering
  - Platform that is "Available" and "Scalable"
  - Cloud Based around virtualization
- Explicit Cost to Use
  - No cost to transfer data, only to use/store
- "Democratization of Infrastructure"
- Rich Data Abstractions
  - Large user data items: blobs
  - Service state: tables
  - Service workflow: queues
  - Simple and Familiar Programming Interfaces
    - REST: HTTP and HTTPS

# Each VM Has…

At Minimum

- CPU: 1.5-1.7 GHz x64
- Memory: 1.7GB
- Network: 100+ Mbps
- Local Storage: 500GB

Up to

- CPU: 8 Cores
- Memory: 14.2 GB
- Local Storage: 2+ TB

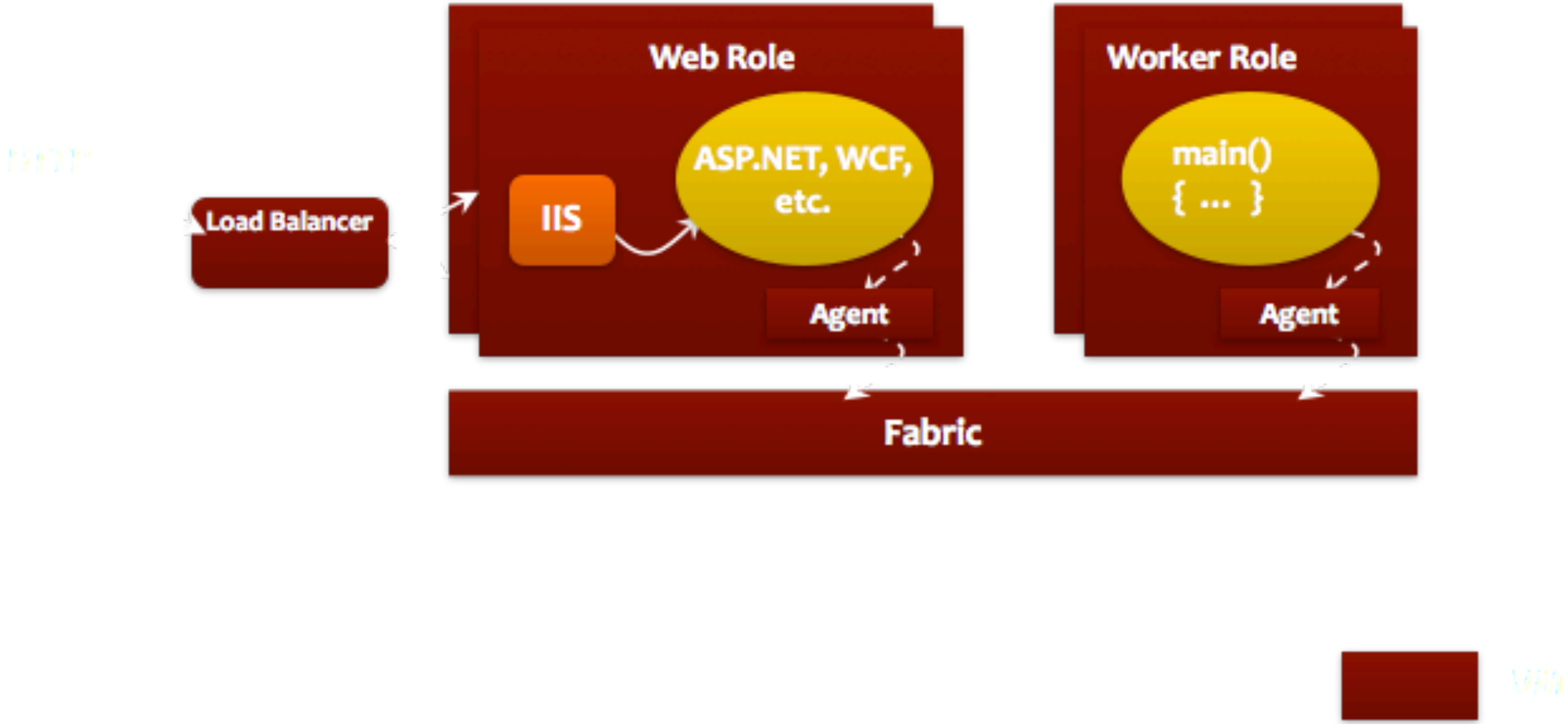
Microsoft® .NET Framework

Windows Server® Internet Information Services 7.0

Windows Server® 2008 Enterprise

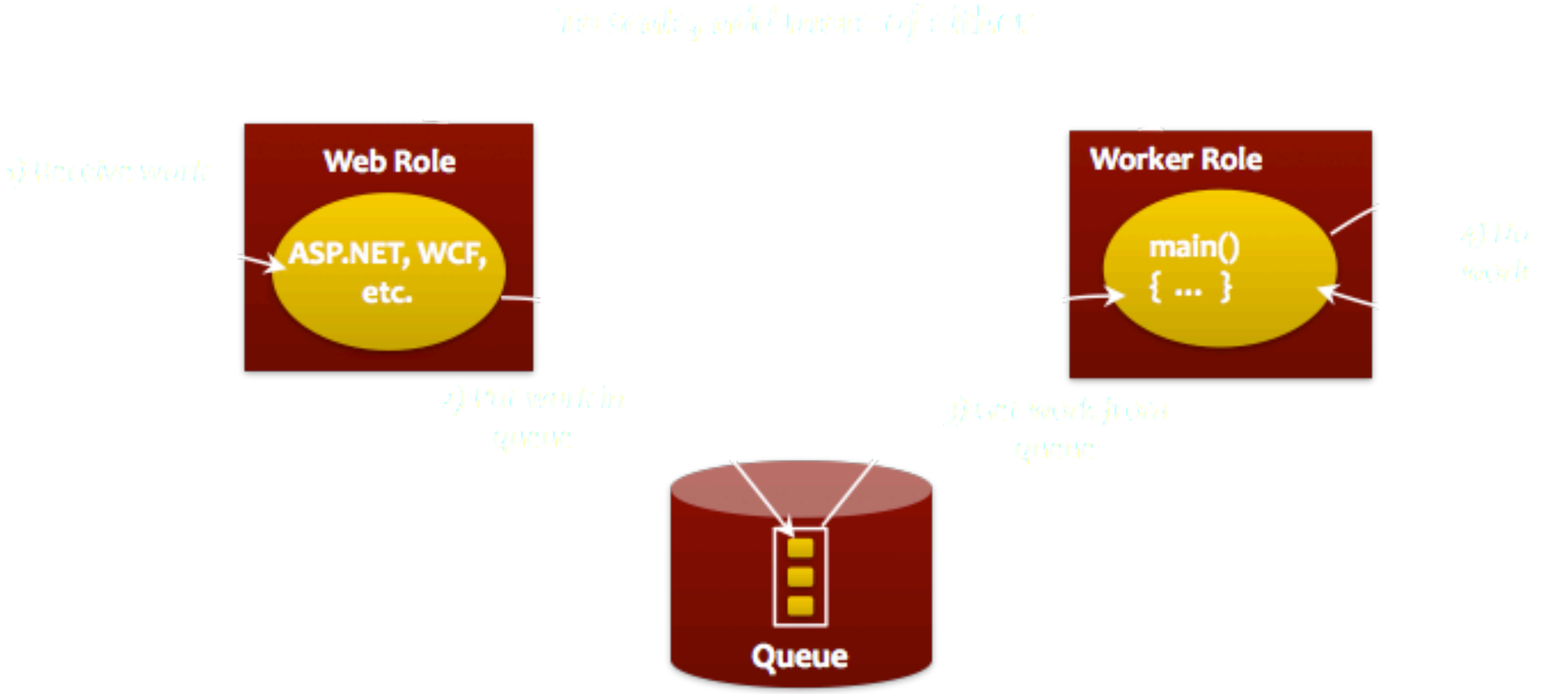
# Windows Azure Compute Service

## A closer look



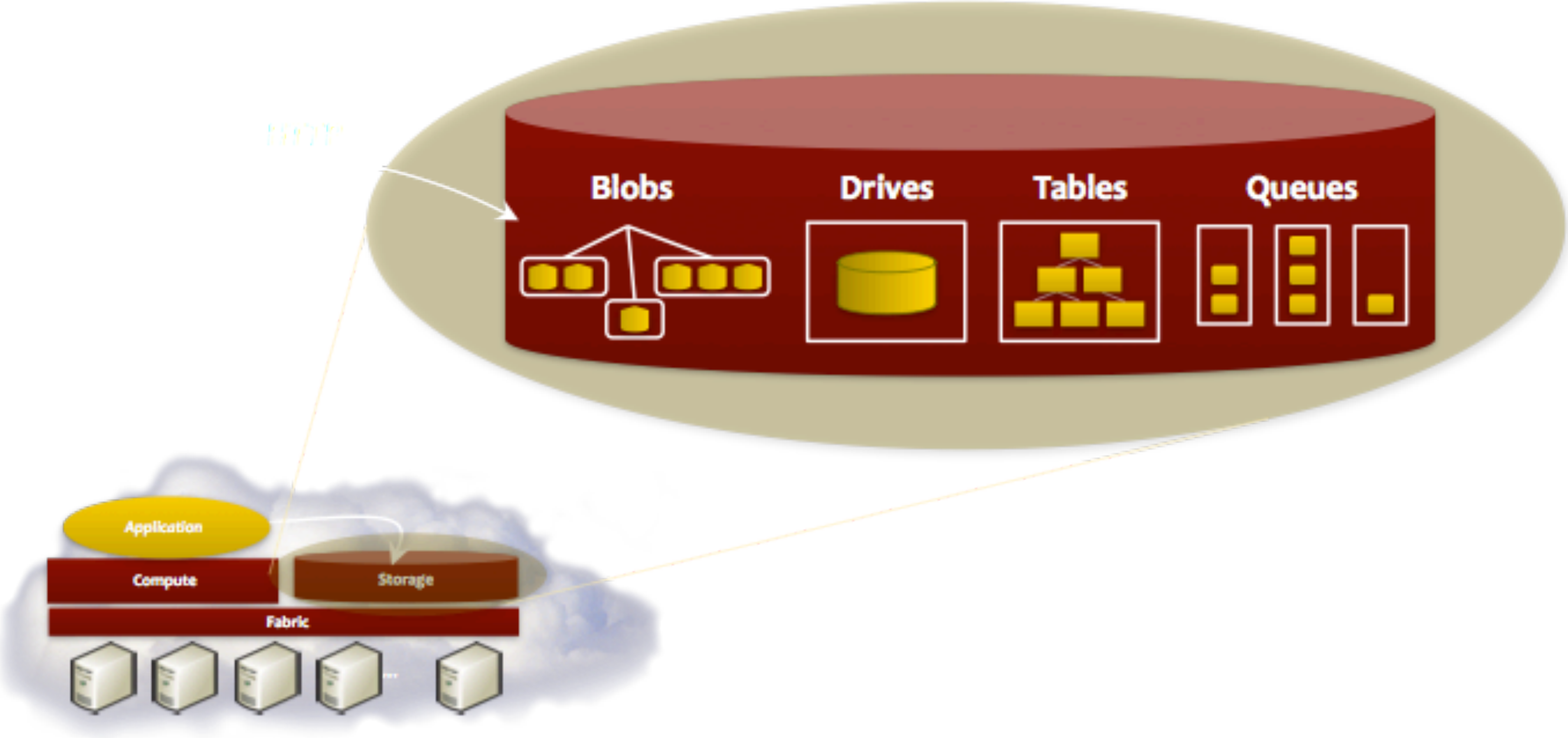
# Suggested Application Model

## Using queues for reliable messaging

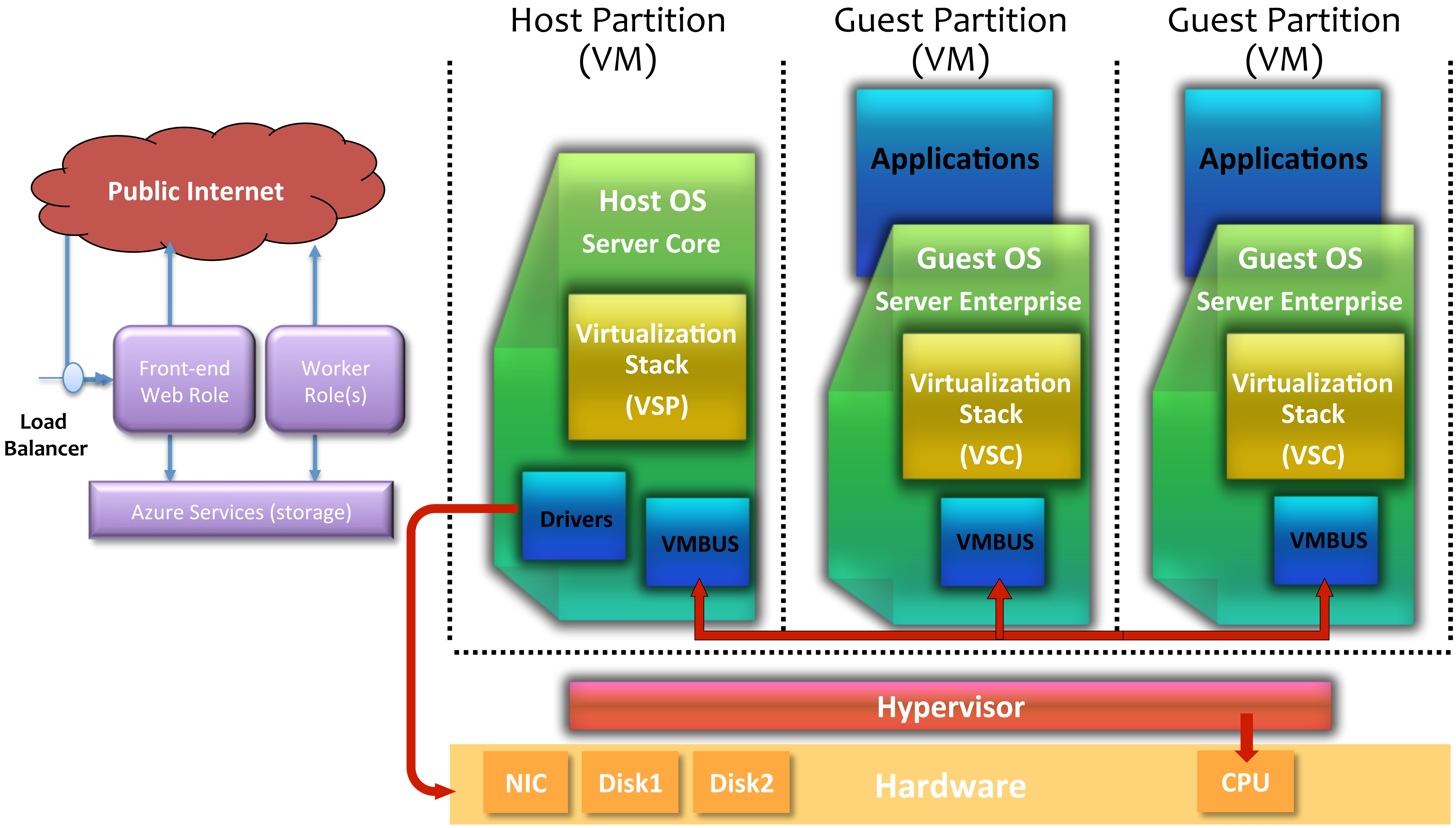


# Azure Storage Service

## A closer look



# Azure Virtualization Architecture



# Module E:
# Distributed Scientific Computing

To Distribute or not to Distribute?

# Distributed Applications Summary

| | Why Distributed? | How Distributed? | Challenges & Issues | How different from \|\| ? |
|---|---|---|---|---|
| Montage | Processing > local limits | Workflow enactor | Coordination | [1, 2] |
| NeKTAR | Processing > local limits (memory) | MPIg | Advanced/Co-reservation | [1?, 4] |
| Ensemble-based/RE | Many compute-intensive task | SAGA, “Advert” | Coordination | [2,3] |
| ClimatePrediction.net | Many small tasks | BOINC, Trickles | Failures, variable # workers | [1, 4] |
| SCOOP | Peak req., naturally, Economic | Customized workflows | Not robust, adv. reservations | [1, 3, 4] |

# “Observations” on Distributed Applications

- Is large (and rich), but the number of effective and extensible DA small
  - More than just submitting jobs here and there!
- Developing DA is a hard undertaking
  - Intrinsic and Extrinsic Factors
  - Unique role of the Execution Environment (Infrastructure)
- Embrace “distributedness”
  - Understanding distributedness,  heterogeneity & dynamic execution is fundamental (e.g., Exascale logically distributed prog. Models)
  - Data-centric application will be the drivers!
- Role for Pattern-oriented and Abstractions-based Development

# Assertion #1: The space of possible DA is large, but number of effective DA small

- Distributed Application: That need multiple resources, or can benefit from the use of multiple resources;
  - .. can benefit from increased peak performance, throughput, reduced time-to-solution
  - More than just HPC or HTC Applications
    - e.g., DDDAS scenarios
- Ability to develop simple or effective distributed applications is limited
  - Applications that utilize multiple resources sequentially, concurrently or asynchronously is low
- Developing DA > just submitting jobs to remote sites!
  - What the pieces of distribution are? How these pieces interact? Flow of information? What is needed to actually deploy and execute the application?

# Assertion #2: Developing DA is a hard undertaking

- Intrinsic reasons why developing DA is fundamentally hard:
  - Control & Coordination over Multiple & Distributed sites
    - Effective coordination in order for whole > sum of the parts
  - Complex design points; wide-range of models of DA
    - Many reasons for using DA, more than (just) peak performance

- Extrinsic:
  - Execution environments will be dynamic, heterogeneous and varying degrees-of-control
    - Fundamental different variation in role of Execution Environment- distinguishing feature of DA from "regular environment" HPC
  - Application types strongly coupled to the infrastructure capabilities, abstractions/tools, & policy:
    - Often development tools assume "specific" deployment and execution environments, or don't where needed!
    - Policies and tools, e.g production DCI has been missing for DDDAS

# Assertion #2: Developing DA is a hard undertaking

- Large number programming systems, tools and environments
  - Lack of  extensible functionality, interfaces & abstractions
    - Interoperability and extensibility become difficult
    - *Art of tool building needs to be more of science!*
- Applications have been brittle and not extensible:
  - Tied to specific tools and/or programming system
  - *Large number of Incomplete Solutions!*
- Unique Role for abstractions for DA and CI
  - Application formulation, development and execution must be less dependent on infrastructure & provisioning details
    - Abstractions for Development, Deployment & Execution
  - A Pattern-Oriented, Abstractions-Based Approach
    - "Abstractions allows innovation at more interesting layers"

# Assertion #3: Embrace Distribution

- “History of computing like pendulum, swings from centralized to distributed”
  - Indications this time there is a fundamental paradigm shift due to DATA
  - Too much to move around; learn how to do analytics/compute *in situ*
- Decoupling and delocalization of the producers-consumers of computation
  - Localized special services; people and collaborations are distributed
- (Ironically) Most applications have been developed to hide from heterogeneity and dynamism; not embrace them
  - Programming models that provide dynamic execution (opposed to static), address heterogeneity etc
  - Logically vs Physically Distributed: NG programming models will need to support dynamic execution, heterogeneity at a logically-distributed level

# Assertion #3: Embrace Distributedness *Corollary: Clouds are not Panacea*

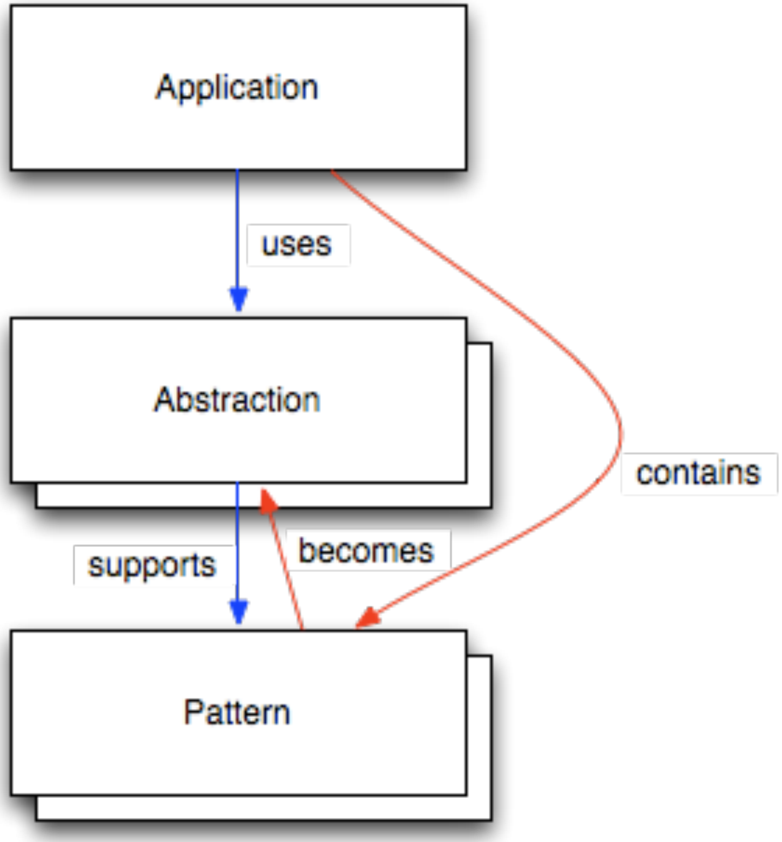
- Clouds: Novel or more of the same?
  - Better control over software environment via virtualization
  - Illusion of unlimited and immediate available resource can lead to better capacity planning and scheduling
    - Partly due to underlying economic model and SLAs

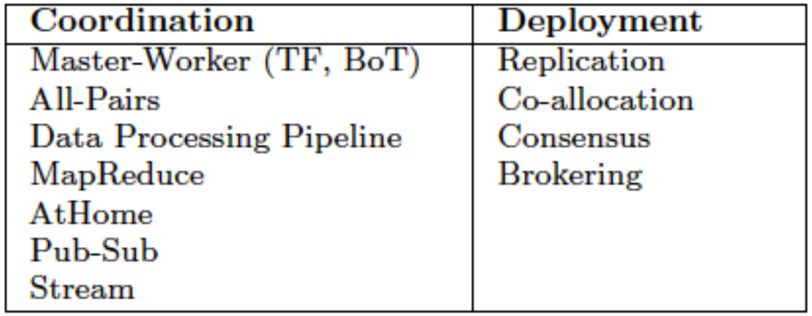
- Clouds do not remove many/all of the challenges inherent in DA
  - Clouds are about provisioning, grids are about federation
  - Fundamental challenges in distribution remain
    - Makes some thing worse as impose a model of strong localization
  - *"The reason why we are so well prepared to handle the multi-core era, is because we took the trouble to understand and learn parallel programming"* – Ken Kennedy

- Clouds part of a larger distributed CI
  - Certain tasks better suited for Grids, others on Clouds

# Assertion #4: Role for a Pattern-Oriented and Abstraction-Based Development Cycle

- Relation between Application, Abstractions and Patterns:
  - Application: Need or can use >1 R
  - Patterns: Formalizations of commonly occurring modes of computation, composition, and/or resource usage
    - Devel, Deploy & Exec Phase
  - Abstractions: Process, mechanism or infrastructure to support a commonly occurring usage

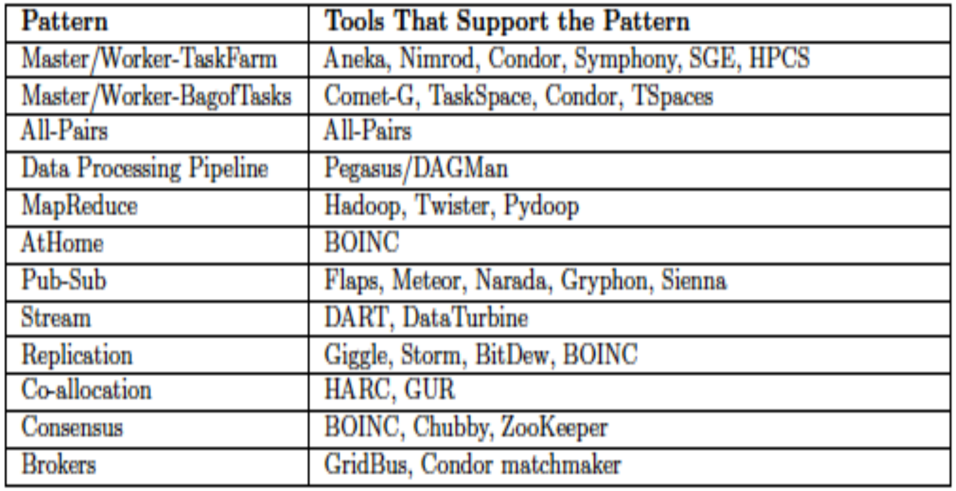




| Coordination | Deployment |
|---|---|
| Master-Worker (TF, BoT) | Replication |
| All-Pairs | Co-allocation |
| Data Processing Pipeline | Consensus |
| MapReduce | Brokering |
| AtHome | |
| Pub-Sub | |
| Stream | |

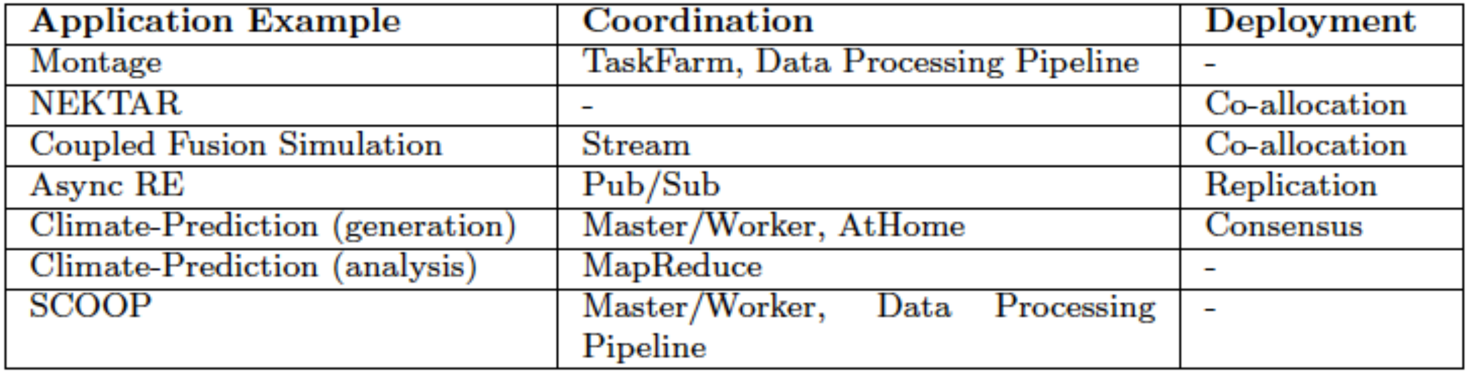
# Assertion #4: Role for a Pattern-Oriented and Abstraction-Based Development Cycle

- Analysis of Distributed Applications leads to three types of patterns
  - Patterns that appear in the Parallel Programming
  - Patterns driven by distributed concerns (eg @HOME, consensus)
  - Patterns addressing distributed environment concerns exclusively (eg co-allocation)



| Pattern | Tools That Support the Pattern |
|---|---|
| Master/Worker-TaskFarm | Aneka, Nimrod, Condor, Symphony, SGE, HPCS |
| Master/Worker-BagofTasks | Comet-G, TaskSpace, Condor, TSpaces |
| All-Pairs | All-Pairs |
| Data Processing Pipeline | Pegasus/DAGMan |
| MapReduce | Hadoop, Twister, Pydoop |
| AtHome | BOINC |
| Pub-Sub | Flaps, Meteor, Narada, Gryphon, Sienna |
| Stream | DART, DataTurbine |
| Replication | Giggle, Storm, BitDew, BOINC |
| Co-allocation | HARC, GUR |
| Consensus | BOINC, Chubby, ZooKeeper |
| Brokers | GridBus, Condor matchmaker |

- There exists tools that support patterns, i.e., provide abstractions



| Application Example | Coordination | Deployment |
|---|---|---|
| Montage | TaskFarm, Data Processing Pipeline | - |
| NEKTAR | - | Co-allocation |
| Coupled Fusion Simulation | Stream | Co-allocation |
| Async RE | Pub/Sub | Replication |
| Climate-Prediction (generation) | Master/Worker, AtHome | Consensus |
| Climate-Prediction (analysis) | MapReduce | - |
| SCOOP | Master/Worker, Data Processing Pipeline | - |

# IDEAS: DA Development Objectives

- **Interoperable:** Ability to work across multiple resources concurrently
  - Includes jobs submission, coordination mechanism,
- **Dynamic:** Beyond legacy static execution & resource allocation models
  - Decisions at both deployment and run-time
  - Dynamical execution is almost fundamental at scale
- **Extensible:** Support new functionality & infrastructure without wholesale refactoring, i.e., lower coupling to tools & infrastructure
- **Adaptive/Autonomic:** Flexible response to fluctuations in dynamic resources, availability of dynamic data
- **Scalable:** Along many dimensions and design points

Challenge: To develop DA effectively and efficiently with IDEAS as first class objectives with simplicity an over-aching concern

# Module E: Project Redux

- Gain sufficient proficiency with SAGA to write a M-W (from scratch) application that uses > 1 XSEDE resource?
- Use Clouds:
  - Can use SAGA to submit jobs to FG-based Clouds?
  - Compare Application X on XSEDE on Clouds?
- …
- *Teamwork is acceptable provided: (i) effort is acknowledged, (ii) clear intellectual contribution from each*

# References

- Python-based Master-Worker:
  - http://pymw.sourceforge.net/
- Google MapReduce
  - http://code.google.com/edu/parallel/mapreduce-tutorial.html
- http://groups.google.com/group/vscse-big-data-for-science-2010/web/course-presentations
- http://futuregrid.org/tutorials

# M-W: Issues to consider

- https://svn.cct.lsu.edu/repos/saga-projects/applications/master_worker
- Aim: Understand trade-off issues along three dimensions:
  - (i) work decomposition (ii) distribution and (iii) coordination
- Homework:
  1. Everything local: For 1 Master and same workload vary: $N_w$ = 2, 4 and 8  Plot times to completion.
  2. With the advert service running remotely, repeat the above. Compare performance with (1)
  3. With the advert service running distributed: Distribute (equally?) the workers across a couple of FutureGrid machines. Compare with (i) and (2)
  4. Extend with user defined "Worker".. use simple worker function.