Programming Patterns and Tools for Cloud Computing

SeSe ACC
Basic service models: IaaS, PaaS, SaaS

Google Docs
Office 365
Dropbox
StochSS

Google AppEngine
OpenShift
CloudFoundry

Amazon Web Services
HP Helion
Backspace
SNIC Cloud

Sam Johnson, http://creativecommons.org/licenses/by-sa/3.0/
Virtual Appliances

Installation of complete software stack packaged as virtual machine images.

- Distributed as files (image formats) or directly in a cloud environment
- Typically used to package up a single application as a VMI, simple distribution
- Common way to make SaaS out of legacy application
- Can easy move between different hosting environments (Public/Private Cloud, VirtualBox, VMWare etc)
- Users typically deploy and use them as virtual private resources

https://en.wikipedia.org/wiki/Virtual_appliance
The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

A major aim of PaaS is to aid in development of “cloudy” applications by:

- Reducing the burden of explicitly managing IaaS
- Cloud interoperability (a goal for some of them)
- Enforcing programming models/frameworks that leads to designs that make good use of the cloud model.
Directly consuming IaaS gives great flexibility and control but may be too complicated for the average user. Platform as a Service aims at simplifying application development by abstracting away IaaS. Also, in some cases, by enforcing certain environments and programming models, the platform can provide added value, particularly autoscaling (see Johan Tordsson’s lectures).
Example: Google App Engine

- Lets you easily build scalable web applications
- Develop in a local SDK sandbox, then deploy to Google Cloud with no code modification
- The platform will autoscale your webservice to meet increased load and traffic
- Supports different languages, e.g. Python, Java, Go and your favorite frameworks like Django, Jinja2 etc.
- But not all possible Python packages (sandbox)
- Works really well for many (traditional) web applications
- Sandbox can be limiting for non-typical applications, like those from CSE
Example: CloudFoundry

http://12factor.net/

https://www.cloudfoundry.org/platform/
Example: OpenShift

https://www.openshift.org/

From RedHat. Similar goals as CloudFoundry, based on Docker and Kubernetes
The capability provided to the consumer is to use the provider’s applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.
Case study: MOLNs and StochSS

www.molns.org, github.com/MOLNs/molns
Cloud virtual appliance/platform for large scale simulations with PyURDME
The tasks: Spatial Stochastic Simulations in Systems Biology

PyURDME ([www.pyurdme.org](http://www.pyurdme.org)) is a modeling and simulation library for stochastic reaction-diffusion simulations of biochemical networks.

- Stochastic process simulated on a computational mesh.
- Applications include studying gene regulation, yeast polarization.
- Output is a time series of spatial data, $X$. 
What problems did we want to solve?

1. PyURDME simulations, in particular parameter sweeps, require large computational resources.
X is just one possible outcome, we need to generate large ensembles of realizations to conduct a statistical analysis.

1k-1M trajectories —> 10*1000 ~ 10Gb data

Falls in the category “High-Throughput Computing (HTC)” and “Many-Task Computing (MTC)”
Potential solution, use HPC - clusters

• We did that for a while, and used them to make large sweeps and write papers ourselves. Then we started to think about the problem of scientific software. There were problems:
  • The modeling process really benefits from *interactivity*, but HPC resources are not for that.
  • It was daunting to try to write software to work with any type of university cluster.
  • The software stack needed is complex and fragile and would likely be very time-consuming to install in the HPC environment (leading to long wait times for users)
  • Computational experiments become hard to reproduce since they rely on specific resources and organizations.
Components in the implementation

**MOLNs**

**molnsclient:** automatic setup and provisioning of a distributed computational environment - the “virtual lab”

**IPython project:** Web based notebook front-end to Python. Sharable, computable documents. Parallel computing in via ipython.parallel.

**molnsutil:** Library for scalable parallel execution of simulation engines. Unified data management in the cloud environments.

**PyURDME:** Spatial stochastic modeling and simulation engine

Simulation engine  Simulation engine
Summary of Experiences so far

- Virtual appliance for building VM images with the software stack for one OS only reduces time to release, critical for a small science team.
- Some parts were easy, essentially just deploying existing software in a virtual environment (e.g. IPython)
- Special care was needed for data transport + Special care was needed for storage abstractions
- As the stack grows, the large monolithic virtual appliance/image is starting to give us problems. We are now looking into alternative technologies as a remedy, e.g. containers and more extensive use of orchestration tools.
What we have not cover in this course that is important for developing SaaS

• Scalable web sites/services
  • frameworks
  • sessions
  • authentication
  • databases
  • security

In practice, this is an integral part of almost all cloud computing software projects, but here many sources of information, tools and frameworks (Webapp2, Django, Flask..), PaaS (GAE...), exist that will make your life fairly easy.
“I need my application to scale”

• What do we mean with a scalable CSE application?
Tightly coupled applications?

- Common in CSE, typical examples are Partial Differential Equation (PDE) solvers

Communication over block-boundaries
HPC programming models

- Message Passing Interface (Distributed Memory)
- OpenMP, Pthreads etc. (Shared Memory)
- CUDA, OpenCL (GPGPU)
- FGPA

- Low-latency interprocess communication is critical to application performance
- Specialized hardware/instruction sets.
- “Every cycle counts”, users of HPC become concerned about VM overhead
- Failed tasks causes major problems/performance losses
- Compute infrastructure organized accordingly
- ’High-Performance Computing, HPC cluster resources. Wait in a queue to obtain dedicated direct access to high-bandwidth hardware resources.
“Cloud computing doesn’t scale”

When you hear those types of statements, they typically refer to conceptions about traditional communication intensive HPC applications, typically implemented with MPI.

- Refers to low-performing network between instances in Public Clouds
- Does not typically account for tailored HPC cluster offerings and placement groups (available in e.g. EC2)

Represents a very narrow view on what applications and programming models are admissible.

StarCluster deploys MPI-clusters over AWS:
http://star.mit.edu/cluster/
Vertical vs. Horizontal Scaling

**Vertical scaling ("scale-up"):**
Increase the capacity of your servers to meet application demand.
- Add more vCPUs
- Add more RAM/storage
- Fundamentally limited by HW capacity of the underlying host
- Cost increases sharply for high-end machines

**Horizontal scaling ("scale-out"):**
Meet increased application demand by adding more servers/storage/etc.
- Also has limitations, but a lot of work going on to improve scale-out properties.
- Cost of adding another machine of the same (commodity) type scales linearly

Typically what we are most interested in in Cloud Computing.
Strong and Weak Scaling

**Strong scaling:** How does the execution time decrease as a function of increased resources, *for a fixed problem size*?

*How quickly can I solve a problem of this size by adding more HW?*
**Weak scaling:** How does the execution time behave as a function of increased resources, *for an increasing problem size*?

How much can I grow my problem size and still maintain the same execution time?
Scalability and Elasticity

Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

Design principles to allow for horizontal scaling and elasticity?

(Discuss/Brainstorm in small groups for 5 min)
Strive for statelessness of task/component

Ideally you want your task to be able to execute anywhere, at any time.

- Robustness
- Can increase, decrease number of workers

But of course, all computations/applications need data and state. It is not so much about if you store it as it is about where you store it.

“Growing is easier than shrinking”
Is the controller machine designed for scalable storage?

Network a potential performance bottleneck

High pressure on critical component (robustness)
Use the object store when you can

S3, Swift

key-value storage, designed for horizontal scaling

Why even store task information on controller? Performance (latency) is one good reason

In this kind of master-slave setup the controller will still maintain a lot of information, route tasks etc.
Use the object store when you can

Decouple the life-time of compute servers and data!

- Needed for elasticity
- Servers are cattle

http://www.slideshare.net/randybias/pets-vs-cattle-the-elastic-cloud-story

- Data can be precious
Another option for robustness, store replicas of data over multiple hosts in a distributed filesystem.

- HDFS (Hadoop, Big Data)
- Does not decouple compute/storage lifetimes as easily
- Designed for extreme I/O throughput
AsynchronousTask

Aim to execute tasks in non-blocking mode, and have as few global synchronization points as possible.
Example: Celery/RabbitMQ (Lab3)

Celery - Distributed Task Queue

key, value store
database
Redis

Our favorite example: MOLNs

PyURDME ([www.pyurdme.org](http://www.pyurdme.org)) is a modeling and simulation library for stochastic reaction-diffusion simulations of biochemical networks.

We have seen how we worked with automation, orchestration etc to make things into cloud service, let’s look a bit at the computational problem.
What problems did we want to solve?

1. PyURDME simulations, in particular parameter sweeps, require large computational resources.

   • Monte Carlo
   • Tasks are numerous
   • But independent

HPC, HTC or MTC?

mostly MTC

But a lot of the time we spend building and debugging models...

or developing solvers

For this, we don’t need big resources, but user interactivity is invaluable.
Interactive Computation

**Personal opinion:** One of the more exciting things about cloud computing technologies from the CSE perspective is the opportunity to enable *interactive parallel* computing.

Non-interactive, obtain complete output when job is done.

```
Input  ---->  Output
```

Interact with partial output as computation proceeds

```
Input

  Output
```

35
MOLNs revisited

**MOLNs**

**molnsclient:** automatic setup and provisioning of a distributed computational environment - the “virtual lab”

**IPython project:** Web based notebook front-end to Python. Sharable, computable documents. Parallel computing in via ipython.parallel.

**molnsutil:** Library for scalable parallel execution of simulation engines. Unified data management in the cloud environments.

**PyURDME:** Spatial stochastic modeling and simulation engine

Simulation engine  
Simulation engine

Amazon EC2  
HP Helion  
OpenStack
Architecture

Diagram showing the architecture with components such as Web Browser, Molns Client, Controller with IPython Notebook Server, Shared Storage, Persistent Storage (S3 / Swift), Workers with IPython Engine(s), and Infrastructure Deployment.
IPython Parallel

Two types of views to your cluster:

- **direct_view**
  - Supports explicit mapping of data/tasks to engines (workers)
  - Run tasks on specific machines
  - MPI
- **load_balanced_view**
  - Asynchronous task queue (uses ZeroMQ)

A set of schedulers provide asynchronous interface to Clients

Engines always block on execution
IPython Parallel

- Let’s you work with many models simultaneously
- Interactively from IPython (Notebook)
- Ideal for rapid prototyping/simple parallel flows

IP parallel in Clouds: Does not attempt to solve the problem of data management

One of the goals of ‘molnsutil’

In molnsutil we have abstractions for storage systems:
- SharedStorage()
- PersistentStorage()
- LocalStorage()
Programming Patterns and Tools for Cloud Computing
Part 2: “Big Data”
What is “Big Data”

Not easy to define just based on size:

• Moving target, today’s “big” might be tomorrows “small”
• Right now, maybe multiple TBs to PBs (maybe)
• Depends on the type of data (structured, unstructured)
• Depends on the “velocity” of the data (streaming data)

Relative to state-of-the art commodity systems:

The problem is “Big Data” if it can’t be stored, handled, processed using commodity systems or traditional RDMS solutions. Typically, single workstations/disks are not sufficient. More about this in Lectures 2 and 3.

The three V’s: Volume, Velocity, Variety
The three Vs, Volume, Velocity and Variety

http://www.datasciencecentral.com/forum/topics/the-3vs-that-define-big-data
Big Data problems involve large datasets, and computations typically become I/O bound:

- Hard Drives capacity to (cheaply) store large amounts of data has increased rapidly but read/write speeds have not.
  
  --> Distribute datasets over multiple disks/machines and process it in parallel.

Poses fundamental requirements on the systems to support such analysis (more about this in Lectures 2,3)
1000 Genomes project

Aim: Create an atlas of the human genetic variation

Find biomedical relevant DNA variations

Understand simple and complex traits, and the in-betweens.

Capture genetic variations that is found in at least 1% of the human population

Today, DNA sequences from ~2600 individuals —> >460 TB or raw data (full genomic sequences), and growing.
Fig. 3. Cloud computing usage in big data.

Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, Samee Ullah Khan

**The rise of “big data” on cloud computing: Review and open research issues**

Information Systems, Volume 47, 2015, 98–115

http://dx.doi.org/10.1016/j.is.2014.07.006
FrameWorks for Big Data

Problems with very large datasets require special frameworks:
- Hadoop
- Hive
- Spark
- ...
- ...

Focus of “Large Datasets for Scientific Applications”
Can be followed as a PhD course. March 28-end of semester.
### Table 3.
Comparison of several big data cloud platforms.

<table>
<thead>
<tr>
<th></th>
<th>Google</th>
<th>Microsoft</th>
<th>Amazon</th>
<th>Cloudera</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Big data storage</strong></td>
<td>Google cloud services</td>
<td>Azure</td>
<td>S3</td>
<td></td>
</tr>
<tr>
<td><strong>MapReduce</strong></td>
<td>AppEngine</td>
<td>Hadoop on Azure</td>
<td>Elastic MapReduce (Hadoop)</td>
<td>MapReduce YARN (Hadoop)</td>
</tr>
<tr>
<td><strong>Big data analytics</strong></td>
<td>BigQuery</td>
<td>Hadoop on Azure</td>
<td>Elastic MapReduce (Hadoop)</td>
<td>Elastic MapReduce (Hadoop)</td>
</tr>
<tr>
<td><strong>Relational database</strong></td>
<td>Cloud SQL</td>
<td>SQL Azure</td>
<td>MySQL or Oracle</td>
<td>MySQL, Oracle, PostgreSQL</td>
</tr>
<tr>
<td><strong>NoSQL database</strong></td>
<td>AppEngine Datastore</td>
<td>Table storage</td>
<td>DynamoDB</td>
<td>Apache Accumulo</td>
</tr>
<tr>
<td><strong>Streaming processing</strong></td>
<td>Search API</td>
<td>Streaminsight</td>
<td>Nothing prepackaged</td>
<td>Apache Spark</td>
</tr>
<tr>
<td><strong>Machine learning</strong></td>
<td>Prediction API</td>
<td>Hadoop+Mahout</td>
<td>Hadoop+Mahout</td>
<td>Hadoop+Oryx</td>
</tr>
<tr>
<td><strong>Data import</strong></td>
<td>Network</td>
<td>Network</td>
<td>Network</td>
<td></td>
</tr>
<tr>
<td><strong>Data sources</strong></td>
<td>A few sample datasets</td>
<td>Windows Azure marketplace</td>
<td>Public Datasets</td>
<td>Public Datasets</td>
</tr>
<tr>
<td><strong>Availability</strong></td>
<td>Some services in private beta</td>
<td>Some services in private beta</td>
<td>Public production</td>
<td>Industries</td>
</tr>
</tbody>
</table>

*Comparison of several big data cloud platforms.*
What is Hadoop?

Original Implementation of MapReduce by Google.
(Read “googlemapreduce.pdf” in the portal)

Hadoop is an OpenSource implementation inspired by Google’s original framework.

First developers: Doug Cutting and Mike Cafarella at that time at Yahoo!, named after Cutting’s son’s toy elephant.
Today, the Hadoop ecosystem is quite large, and contains many analytics tools. The core components are:

- HDFS (Distributed Filesystem)
- Hadoop commons (libraries, utilities etc.)
- YARN (Resource management platform, not in 1.2.1)
- MapReduce
What is MapReduce?

• A programming model for writing (simple) (data)parallel programs
• A framework to execute such programs.

Advantages:
If you manage to express your problem as MapReduce, the framework handles:

• Data distribution (HDFS)
• Fault-tolerance (on many levels)
• Brings computations close to data*
• Runs on commodity hardware
• Scales to 1000’s of nodes (failure is inevitable)
• Simple programming model (when you know it...)

The data (and the cluster) should be very large for the advantages to become apparent.

It was designed for the case of commodity HW without high performance network connects. You should not think of HDFS as a “HPC parallel filesystem”. You don’t communicate data to compute, you communicate compute to data.
Basic design

Handles Jobs, Tasks, logic of computing close to data

Distributed Filesystem, HDFS

http://en.wikipedia.org/wiki/Apache_Hadoop
There are a couple of excellent basic tutorials for getting started to playing with Hadoop on single node clusters or small distributed clusters (yes, this is fun, I recommend it if you have the time and a some Linux (admin) experience).

http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/


Also, it is rather easy to set up a “toy-hadoop” on your own laptop:

https://hadoop.apache.org/docs/r1.2.1/single_node_setup.html
Setting up Hadoop in Production

This is a complex process. In practice, there are many vendors that provide hosted services, or projects to make it simpler e.g.

- Cloudera
- Hortonworks
- Elastic MapReduce (EMR) in AWS
- OpenStack Sahara
- Apache Bigtop

--> More important to get familiar with MapReduce.
So how do you use it?

https://developers.google.com/appengine/docs/python/dataprocessing/
You provide.

Framework provides optimized sort, shuffle.
Key stages of MapReduce

1. Split input
2. Map
3. Shuffle
4. Reduce
Hadoop Streaming

What if I don’t want to use Java?

The Hadoop streaming API let’s you write the mapper and reducer as a stand alone program, reading from stdin and writing to stdout. You mapper and reducer can now be written using Python, Ruby, Bash shell, R, you name it.

Logically, it corresponds to the following Unix pipe:

```
cat input.txt | ./map | sort | ./reduce > output.txt
```
Performance guidelines

- There is a significant latency in starting Map-tasks by the framework
- Maps should do considerable work (rule of thumb, work at least 1min)
- Default block size is 64-128MB, input files should be large (GB sized)
- Works best for “one-shot” computations, not iterative flows, due to the large latency.
- Use Combiner (local reduce) to reduce the network overhead for shuffle/reduce.
Criticism against MapReduce

- Restricted programming model.
- Performance
- Not novel

An old but still interesting discussion:

http://homes.cs.washington.edu/~billhowe/mapreduce_a_major_step_backwards.html

http://blog.tonybain.com/tony_bain/2010/09/was-stonebraker-right.html
Apache Spark

Spark™ is a fast and general engine for large-scale data processing.

**Speed**
Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.

**Ease of Use**
Write applications quickly in Java, Scala, Python, R.

Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it interactively from the Scala, Python and R shells.

**Generality**
Combine SQL, streaming, and complex analytics.

Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.