Look Up!

Your Future is in the Cloud

James Larus
Microsoft Research

PLDI, June 2013
What is the Cloud?

Data centers at scale ↔ networked elastic computation ↔ big data
Paradigm Shift

Single computer to clusters + mobile
“Batch” and “Interactive” Cloud Computing

Dean, Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, CACM 1/08.

DeCandia, et al., Dynamo: Amazon’s Highly Available Key-value Store, SOSP 11/07.
Cloud Computing Buzzwords

"IaaS" Infrastructure-as-a-Service
host

"PaaS" Platform-as-a-Service
build

"SaaS" Software-as-a-Service
consume

http://www.silverlighthack.com/
Driving Factors

- Mobile computing
- Economies of scale
- Elastic computing
- End of Moore’s Law
- Too much data
Globally, mobile data traffic will grow 13-fold from 2012 to 2017, a compound annual growth rate of 66%.

Globally, mobile data traffic will reach 11,156,995 Terabytes (11.16 Exabytes) per month in 2017, the equivalent of 2,789 million DVDs each month or 30,746 million text messages each second.

Global mobile data traffic will grow 2.8 times faster than Global fixed IP traffic from 2012 to 2017.

Globally, mobile data traffic in 2017 will be equivalent to 771x the volume of Global mobile traffic ten years earlier (in 2007).

Globally, mobile traffic per user will reach 2,037 megabytes per month in 2017, up from 201 megabytes per month in 2012, a CAGR of 56%.
Economies of Scale of Large Datacenters

2006 comparison of very large service with mid-size: (~1000 servers):

- Large Service [[$13/\text{Mb/s/mth}]]: $0.04/\text{GB}$
  Medium [[$95/\text{Mb/s/mth}]]: $0.30/\text{GB}$ (7.1x)

- Large Service: $4.6/\text{GB/year}$ (2x in 2 Datacenters)
  Medium: $26.00/\text{GB/year}$* (5.7x)

- Large Service: Over 1,000 servers/admin
  Enterprise: ~140 servers/admin (7.1x)

*Cost estimates based on historical data and industry standards.

Source: Microsoft.

James Hamilton, Cloud Computing Economies of Scale, MIX 10
Elastic Computing
Moore’s “Law” and Limiting Exponentials ...

The experts look ahead

Cramming more components onto integrated circuits

With unit cost falling as the number of components per circuit rises, by 1975 economics may dictate squeezing as many as 65,000 components on a single silicon chip

By Gordon E. Moore

Director, Research and Development Laboratories, Fairchild Semiconductor division of Fairchild Camera and Instrument Corp.

1975

Intel 4004

Intel Core i7

EUV

Trouble
$1.2 \times 10^{21}$ New Bytes of Information in 2010

Source: IDC, as reported in The Economist, Feb 25, 2010
Economics of Storage

Disk Storage (per gigabyte) $0.04
Web Storage (per gigabyte) $1.02

... free storage is like free puppies ...

Source: Wired Magazine April 2010; Figures represented in USD
In 2000 the Sloan Digital Sky Survey collected more data in its 1st week than was collected in the entire history of Astronomy.

By 2016 the New Large Synoptic Survey Telescope in Chile will acquire 140 terabytes in 5 days - more than Sloan acquired in 10 years.

The Large Hadron Collider at CERN generates 40 terabytes of data every second.
Genetics Gets Personal

3e09 bytes per person x 6e09 people = 1.8e19 bytes = 10K petabytes

(Why stop at 1 cell per person? Why stop at humans? Why stop at animals?)

Source: George Church, Harvard Medical School, as reported in IEEE Spectrum, Feb ‘10. Figures represented in USD
Move Computation to Data(center)

Never underestimate the bandwidth of a FedEx Truck full of disks hurtling down the highway.

— Jim Gray, 2003

5000 2TB drives = 10 petabytes (10e16)
Next day delivery ≈ 10e6 seconds
1e10 bytes/sec ≈ 10 GB/sec ≈ 100 Gb/sec
New World, Still Has Programming Problems

Familiar problems

- Pervasive parallelism
- Partial failure
- High, variable communication latency
- Replication for reliability and throughput
- Deadlines and approximate computation

New challenges

- Multitenacy
- Services
- Involuntary upgrades
- Reliability, availability
- Impact = people * rate
Pervasive Parallelism
“A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable” – Leslie Lamport
High and Variable Communication Latency
Replication for Reliability and Throughput

Eric Brewer’s CAP Theorem:
Consistency
Availability
Partition Tolerance

Choose 2
Deadlines and Approximate Computation

He, Y., et al. Zeta: Scheduling Interactive Services with Partial Execution. SOCC, 10/12.
Multitenancy – Shared Service

Single Tenant

- Tenant
  - Application
  - App Platform
  - DB
  - OS

Multi Tenant

- Tenant
  - Application
  - App Platform
  - DB
  - OS
Services Must Be Reliable and Available
What Can Programming Languages Do?

First-class language support

- Shared memory and message passing
- Replicated data
- Failure handling
- Domain-specific languages

Massively parallel development tools

- Failure detection and notification at scale
- Correctness and performance debugging for services

Services life cycle support

- Strong versioning
- Live update
- Introspective monitoring and control
Existing Languages

Many languages in use, no focus on Cloud
Programming Software-Defined Networks

High-level language for programming OpenFlow networks

Paper: Wed, 4pm!
Orleans

Framework for productive cloud software development
Experimental .NET library from Microsoft Research
Runs on desktops, servers, Microsoft Azure
Used by Microsoft for several services

Radically simplified, prescriptive cloud programming model
Actors
Asynchronous messaging
Lightweight transactions
Persistence
Adaptive performance management

Burden of correctness and performance on Orleans (not dev)
Visual Basic of cloud programming
Orleans Features

- Grains
- Grain activations
- Messages
- Promises
- Transactions
- Adaptive performance
- Persistence
Actor Based

**Customer Grain**

- **Field** | **Value**
  - Name | “John Doe”
  - Email | “john.doe@hotmail.com”
  - Address | “123 Main St., Anywhere UR 01234”

**Grain (actor)**

- **Methods**
  - Checkout
  - AddProduct
  - RemoveProduct

**Message Queue**

**State**

**Grain ID_1**

**Grain ID_2**
Why Actors?

Fine-grain distributed objects

Widely used, natural abstraction: computation as reusable service
Isolation and message passing mirror physical hardware

Secure and isolated computation with clear communications

Singularity OS
Computation replication

Encapsulated and partitioned data

Scalability and replication

Natural integration with persistent storage

Grain resides on disk until activated
In-Grain Programming Model

```
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>&quot;John Doe&quot;</td>
</tr>
<tr>
<td>Email</td>
<td>&quot;<a href="mailto:john.doe@hotmail.com">john.doe@hotmail.com</a>&quot;</td>
</tr>
<tr>
<td>Address</td>
<td>&quot;123 Main St., Anywhere UR 01234&quot;</td>
</tr>
<tr>
<td>Products</td>
<td></td>
</tr>
</tbody>
</table>

Grain ID_1
Grain ID_2
Checkout
AddProduct
RemoveProduct
```
Radically Simple Parallel Programming

Minimize shared-memory parallelism
Parallel processes, not threads
(Can thread single task when valuable)

Explicit asynchrony
All communications is asynchronous
Error-propagation and handling are first-class constructs

Simplify challenging aspects of distributed systems
Force developers to use restricted, less error-prone parallelism
Discourage synchronous constructs that lead to poor scalability and performance

Enable automatic grain replication
Transparent performance improvement
Promises - Resolve Impedance Mismatch

Call/return is synchronous, message is asynchronous

Bind remote computation to processing of eventual result
  Communication operation returns a promise
  Caller binds promise to closure
    Evaluated when result becomes available
    Produces promise for result of closure, and so on...

Explicit representation of concurrency
  Compose stages via dataflow dependencies

Errors propagated
  Error handling can be added where needed – ‘asynchronous try/catch’

(Replaced by .NET 4.5 async construct)
Replication

Scalability comes from partitioning and replication
  Grains encourage service model

Concurrent grain activations handle independent requests
  Orleans runtime dynamically replicates grains to handle load

Inconsistent can occur in shared, persistent state
  Transactions and state merging can approximate consistency at reasonable cost
State Merging

Default: “last writer wins”
Eventual consistency

Library of commutative replicated data structures
(Marc Shapiro, INRIA)
Set, tree, hashtable
Consistent semantics across distributed changes

Revisions and isolation types
(Burckhardt, Leijen, MSR)
Branch at activation
Merge activations’ updates at persistent store
Propagate updates to other activations

Burckhardt, Baldassin, Leijen Concurrent Programming with Revisions and Isolation Types, OOPSLA 2010.
Transactions

ACID

Isolation

Atomicity

Consistency
Limited Transactions

**Isolation**: grain activations are isolated from activations responding to other requests

Appears as if system is processing only one request

**Atomicity**: computation either completes successfully, or no persistent state changes

If computation fails, can re-execute original request

**Consistency**: only one activation of grain allowed in transaction

Multiple messages see consistent view of grain’s state

**Not serializable!**

Local properties (atomic writes), but no global guarantees across transactions
Challenges

Distributed transactions are notoriously difficult and expensive

Use simple, non-serializable transactions

Avoid single point of conflict

Experimental: still needs refinement
Adaptive Performance Management

Grain Placement

Grain Migration
Automatic Performance Tuning

Measure performance of grain
  # of requests, latency, throughput

Create more activations if grain is heavily loaded

Shift load from overloaded servers by moving activations

Transparent to application
  Possible because of location-independent grain programing model
Experience

Intended for “Mort”, used by “Einstein”

Don’t know if it is Cloud VB

Developers like simplicity and predictability

Easy to get started
Hides complexity of underlying systems
Focus on architecture and application, not system details
Predictable performance behavior
Conclusion

Computing eras: mainframe, mini, pc, mobile, cloud

Punctuated equilibrium (Stephen Jay Gould)

Disruptive change ⇔ research opportunity

Language & compiler community

Many programming challenges
Strong need for better languages and tools
Do not miss the cloud opportunity
Appropriate Size of a Grain

Reducing grain overhead enables finer-grain objects

Share resources (communication channels and OS threads) among grains in silo (process)

- **Chirper**
  - Simplified Twitter-like system
  - ~200 lines of application code

- **Horton**
  - Distributed graph database

- **Sparse Linear Algebra**
  - Eigenvectors of large matrices (1B x 1B)
  - PageRank calculation

Graph partition

User Message Address book

Processor
Persistence