

# Distributed Data Parallel Computing: The Sector Perspective on Big Data

**Robert Grossman**

July 25, 2010

Laboratory for Advanced Computing  
University of Illinois at Chicago

Open Data Group

Institute for Genomics & Systems Biology  
University of Chicago 1



LABORATORY FOR  
ADVANCED COMPUTING



open data

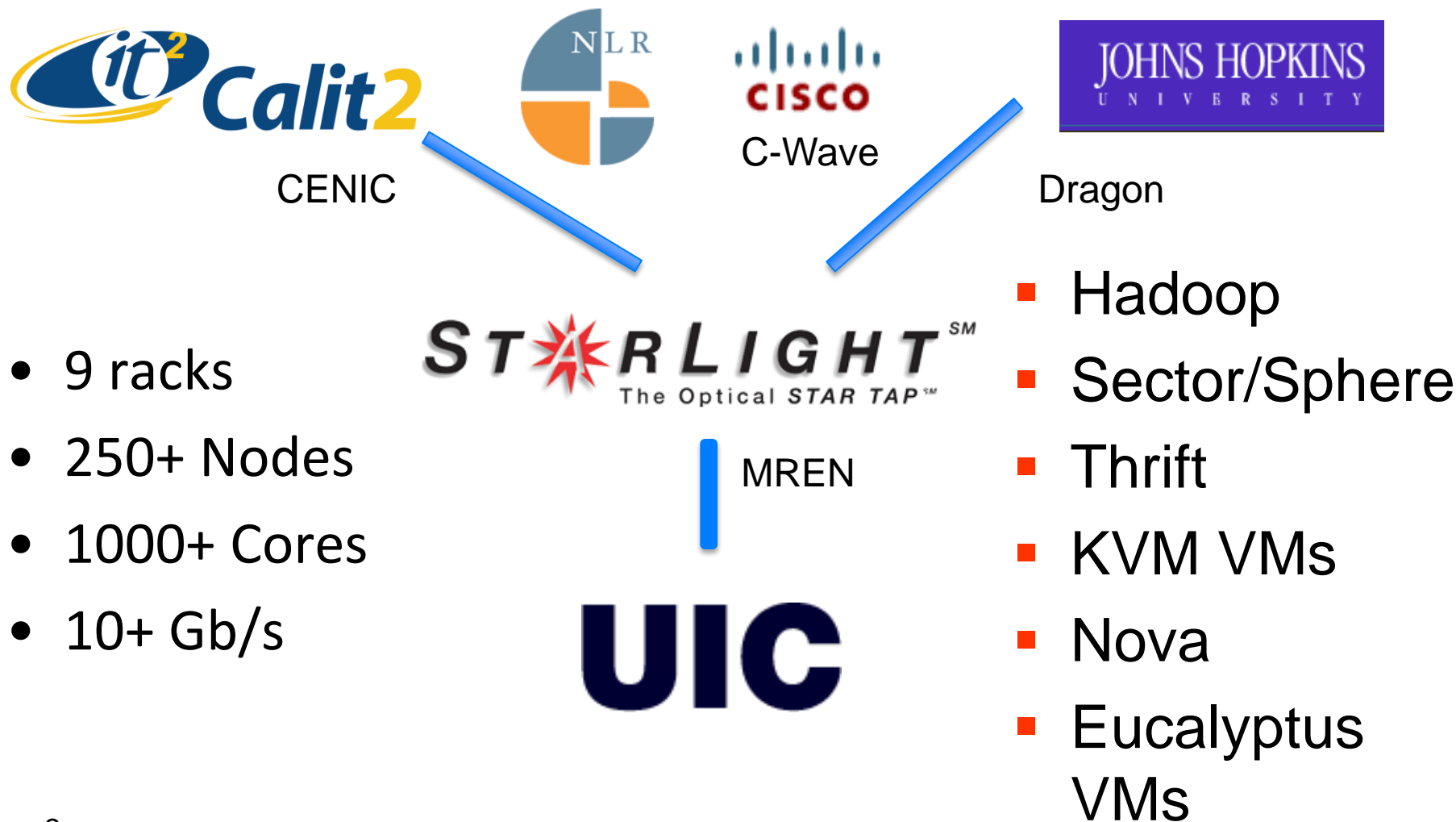


Institute for  
Genomics &  
Systems Biology

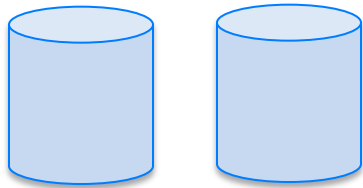
# Part 1.



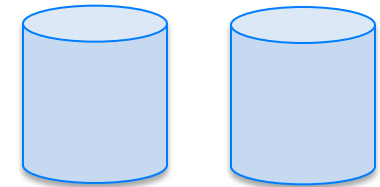
# Open Cloud Testbed



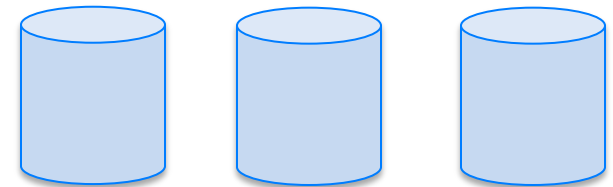
# Open Science Data Cloud



NSF OSDC PIRE  
Project – Working  
with 5 international  
partners (all  
connected with 10  
Gbps networks).

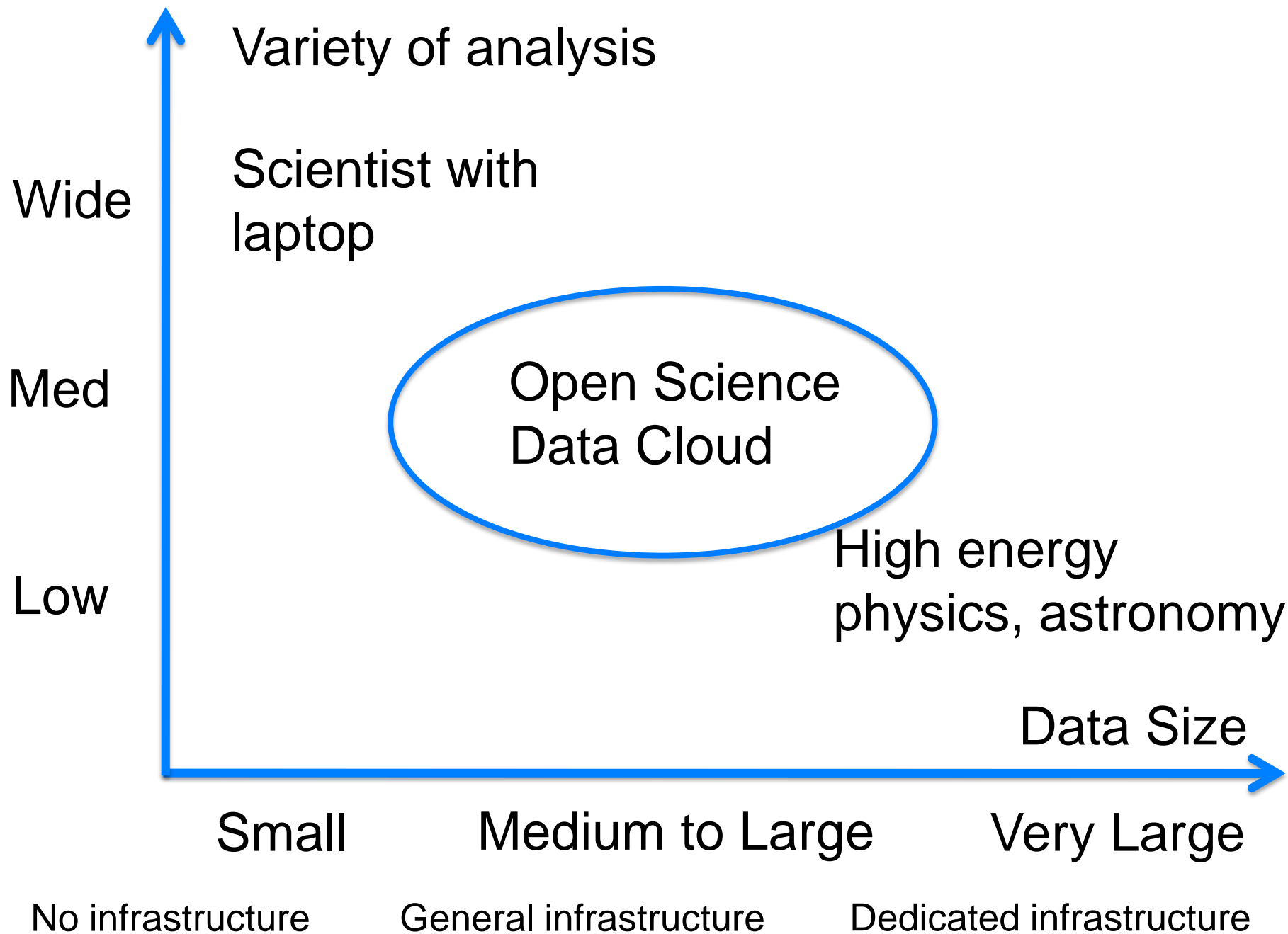


sky cloud



Bionimbus (biology &  
health care)





## Part 2

# What's Different About Data Center Computing?





Data center scale computing provides storage and computational resources at the scale and with the reliability of a data center.

# **The Datacenter as a Computer**

*An Introduction to the Design of  
Warehouse-Scale Machines*

Luiz André Barroso and Urs Hölzle  
Google Inc.

A very nice recent book by  
Barroso and Holzle

*SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE # 6*



MORGAN & CLAYPOOL PUBLISHERS



# Scale is new



# Elastic, Usage Based Pricing Is New



costs the same as

1 computer in a rack  
for 120 hours



120 computers in three  
racks for 1 hour

# Simplicity of the Parallel Programming Framework is New

A new programmer can develop a program to process a container full of data with less than day of training using MapReduce.



## Elastic Clouds



Goal: Minimize cost of virtualized machines & provide on-demand.

## Large Data Clouds



Goal: Maximize data (with matching compute) and control cost.

## HPC



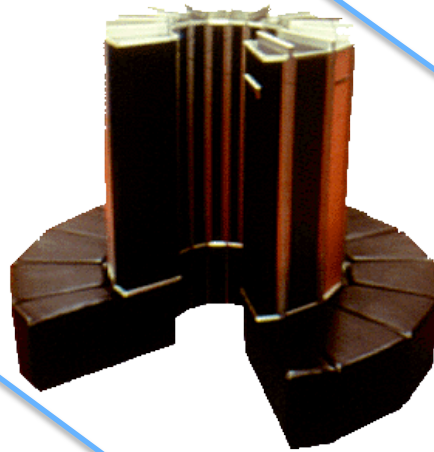
Goal: Minimize latency and control heat.

2003  
10x-100x



data  
science

1976  
10x-100x



simulation  
science

1670  
250x



experimental  
science

1609  
30x

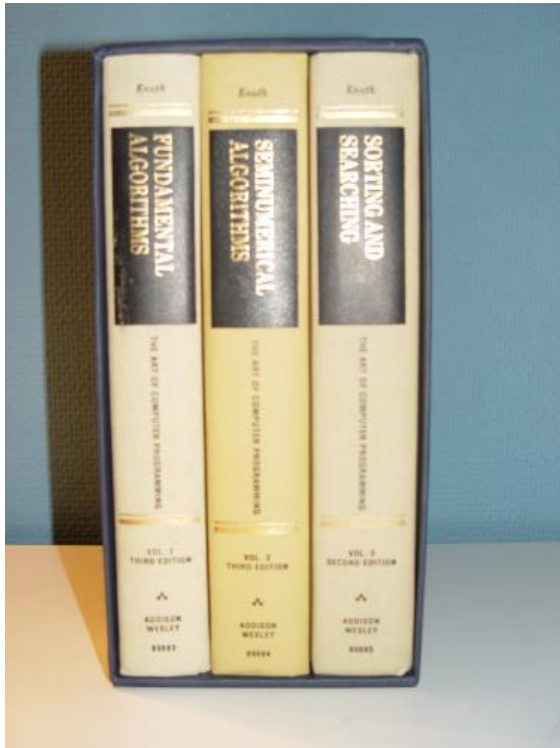


	<b>Databases</b>	<b>Data Clouds</b>
Scalability	100's TB	100's PB
Functionality	Full SQL-based queries, including joins	Single keys
Optimized	Databases optimized for safe writes	Data clouds optimized for efficient reads
Consistency model	ACID (Atomicity, Consistency, Isolation & Durability)	Eventual consistency
Parallelism	Difficult because of ACID model; shared nothing is possible	Parallelism over commodity components
Scale	Racks	Data center

	<b>Grids</b>	<b>Clouds</b>
Problem	Too few cycles	Too many users & too much data
Infrastructure	Clusters and supercomputers	Data centers
Architecture	Federated Virtual Organization	Hosted Organization
Programming Model	Powerful, but difficult to use	Not as powerful, but easy to use

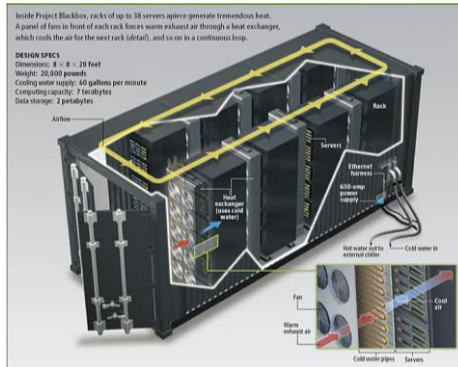
# Part 3

## How Do You Program A Data Center?





# How Do You Build A Data Center?

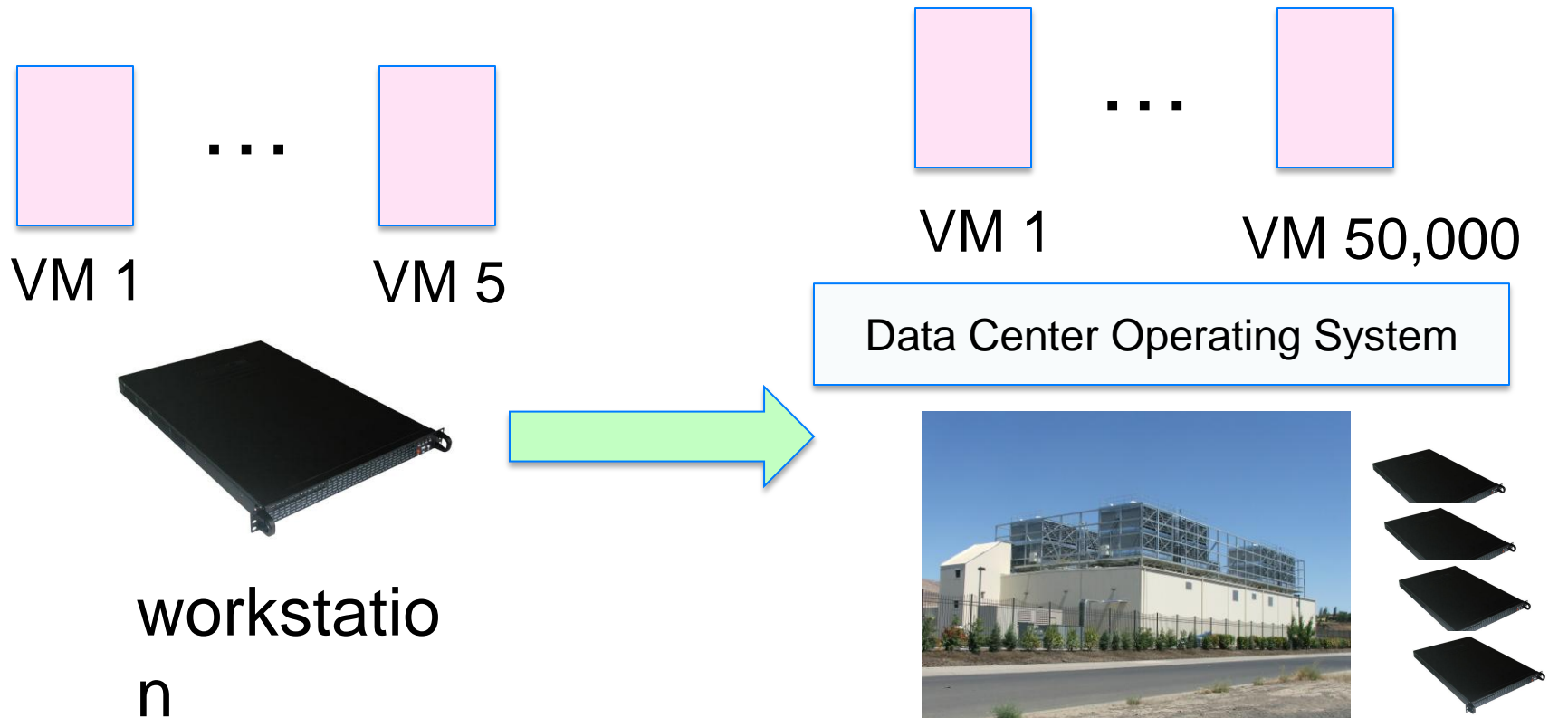


- Containers used by Google, Microsoft & others
- Data center consists of 10-60+ containers.



Microsoft Data Center, Northlake, Illinois

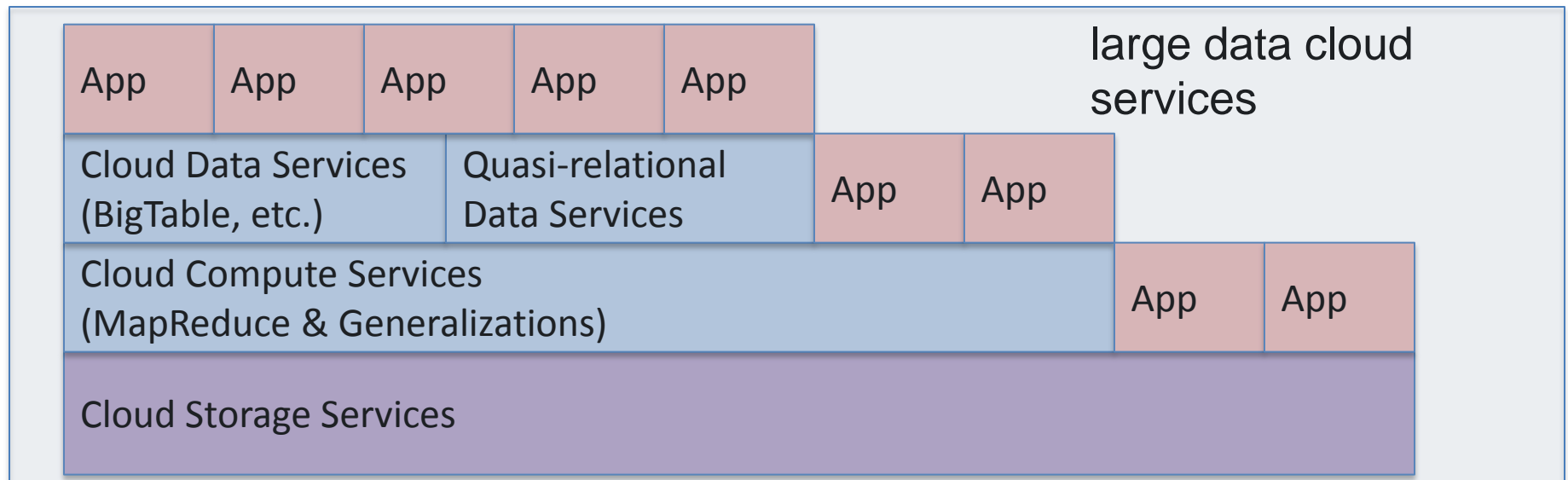
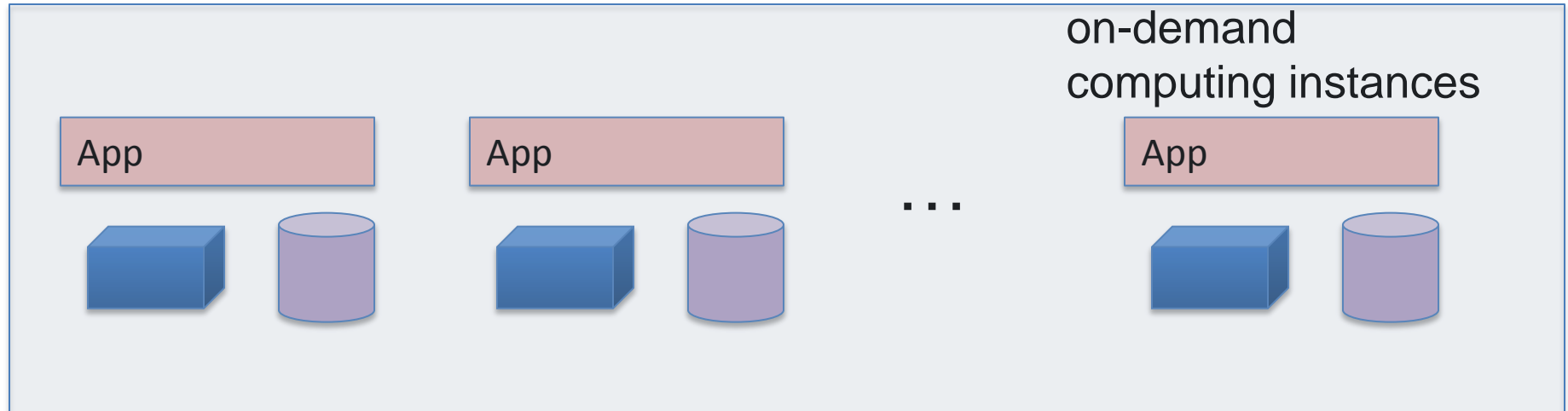
# What is the Operating System?



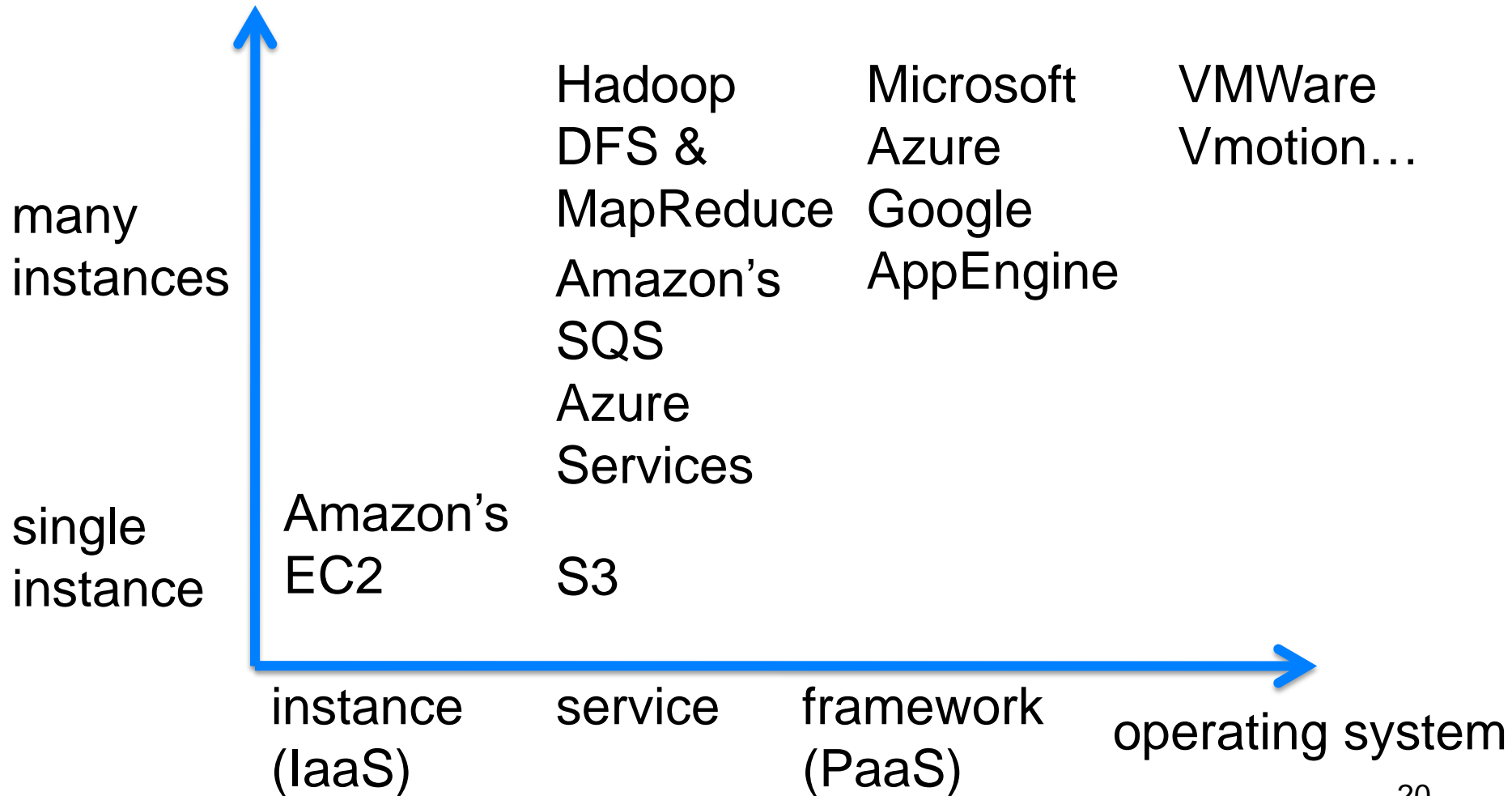
- Data center services include: VM management services, VM fail over and restart, security services, power management services, etc.

# Architectural Models:

## How Do You Fill a Data Center?



# Instances, Services & Frameworks



# Some Programming Models for Data Centers

- Operations over data center of disks
  - MapReduce (“string-based”)
  - Iterate MapReduce (Twister)
  - DryadLINQ
  - User-Defined Functions (UDFs) over data center
  - SQL and Quasi-SQL over data center
  - Data analysis / statistics functions over data center

# More Programming Models

- Operations over data center of memory
  - Memcached (distributed in-memory key-value store)
  - Grep over distributed memory
  - UDFs over distributed memory
  - SQL and Quasi-SQL over distributed memory
  - Data analysis / statistics over distributed memory

# Part 4. Stacks for Big Data



**Sector/Sphere**



**CLOUDSTORE**

**Project Voldemort**  
*A distributed database.*

# The Google Data Stack

## The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung  
Google

### ABSTRACT

We have designed and implemented the Google File System (GFS), a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients. While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environments, both current and anticipated, that reflect a marked departure from those of earlier file system designs. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a correspondingly number of client machines. The quantity and quality of the components visually guarantee that some are not functional at any given time and some will not recover from their current failures. We have seen problems caused by application bugs, operating system bugs, hardware errors, and the failure of disks, memory, controllers, networking, and power supplies. Therefore, constant monitoring, error detection, fault tolerance, and automatic recovery must be integral to the system.

Second, files are large by traditional standards. Multi-GB files are common. Each file typically contains many application objects such as web documents. When we are regularly working with fast growing data sets of easily Tera comprising billions of objects, it is unworkable to manage billions of approximately 1KB-sized files even when the file system could support it. As a result, design assumptions and parameters such as I/O operation and block sizes have to be avoided. Third, most files are mutated by appending new data rather than overwriting existing data. Random writes within a file are practically non-existent. Once written, the file is only read, and only very occasionally. A variety of data usage characteristics. Some may constitute large repositories that data analysis programs scan through. Some may be data streams continuously generated by running applications. Some may be archival data. Some may be intermediate results produced on one machine and processed on another, whether simultaneously or later in time. Given this access pattern on large files, depending between the focus of performance optimization and storage systems, while caching data blocks in the client lowers its appeal.

Fourth, redesigning the applications and the file systems API benefits the overall system by increasing our flexibility.

Over the past five years, the authors and many others at Google have implemented hundreds of special-purpose computations that process large amounts of raw data, such as crawled documents, web request logs, etc., to compute various kinds of derived data, such as inverted indices, various representations of the graph structure of web documents, summaries of the number of pages crawled per host, the set of most frequent queries in a

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key-value pairs, and then applying a *reduce* operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user-specified *map* and *reduce* operations allows us to parallelize large computations easily and to use *re-execution* as the primary mechanism for fault tolerance.

The major contributions of this work are a simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs.

Section 2 describes the basic programming model and gives several examples. Section 3 describes an implementation of the MapReduce interface tailored towards our cluster-based computing environment. Section 4 describes several refinements of the programming model that we have found useful. Section 5 has performance measurements of our implementation for a variety of tasks. Section 6 explores the use of MapReduce within Google including our experiences in using it as the basis

for the design and implementation of Bigtable.

Section 7 describes the data model in more detail, and Section 8 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 9 describes the fundamentals of the Bigtable implementation, and Section 10 describes some of the refinements that we made to improve Bigtable's performance. Section 11 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 12, and discuss some lessons we learned in designing and supporting Bigtable in Section 13. Finally, Section 10 describes related work, and Section 11 presents our conclusions.

Section 2 describes the data model in more detail, and Section 3 provides an overview of the client API. Section 4 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 5 describes the fundamentals of the Bigtable implementation, and Section 6 describes some of the refinements that we made to improve Bigtable's performance. Section 7 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 8, and discuss some lessons we learned in designing and supporting Bigtable in Section 9. Finally, Section 10 describes related work, and Section 11 presents our conclusions.

Section 12 describes the data model in more detail, and Section 13 provides an overview of the client API. Section 14 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 15 describes the fundamentals of the Bigtable implementation, and Section 16 describes some of the refinements that we made to improve Bigtable's performance. Section 17 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 18, and discuss some lessons we learned in designing and supporting Bigtable in Section 19. Finally, Section 20 describes related work, and Section 21 presents our conclusions.

Section 22 describes the data model in more detail, and Section 23 provides an overview of the client API. Section 24 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 25 describes the fundamentals of the Bigtable implementation, and Section 26 describes some of the refinements that we made to improve Bigtable's performance. Section 27 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 28, and discuss some lessons we learned in designing and supporting Bigtable in Section 29. Finally, Section 30 describes related work, and Section 31 presents our conclusions.

Section 32 describes the data model in more detail, and Section 33 provides an overview of the client API. Section 34 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 35 describes the fundamentals of the Bigtable implementation, and Section 36 describes some of the refinements that we made to improve Bigtable's performance. Section 37 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 38, and discuss some lessons we learned in designing and supporting Bigtable in Section 39. Finally, Section 40 describes related work, and Section 41 presents our conclusions.

Section 42 describes the data model in more detail, and Section 43 provides an overview of the client API. Section 44 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 45 describes the fundamentals of the Bigtable implementation, and Section 46 describes some of the refinements that we made to improve Bigtable's performance. Section 47 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 48, and discuss some lessons we learned in designing and supporting Bigtable in Section 49. Finally, Section 50 describes related work, and Section 51 presents our conclusions.

Section 52 describes the data model in more detail, and Section 53 provides an overview of the client API. Section 54 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 55 describes the fundamentals of the Bigtable implementation, and Section 56 describes some of the refinements that we made to improve Bigtable's performance. Section 57 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 58, and discuss some lessons we learned in designing and supporting Bigtable in Section 59. Finally, Section 60 describes related work, and Section 61 presents our conclusions.

Section 62 describes the data model in more detail, and Section 63 provides an overview of the client API. Section 64 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 65 describes the fundamentals of the Bigtable implementation, and Section 66 describes some of the refinements that we made to improve Bigtable's performance. Section 67 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 68, and discuss some lessons we learned in designing and supporting Bigtable in Section 69. Finally, Section 70 describes related work, and Section 71 presents our conclusions.

Section 72 describes the data model in more detail, and Section 73 provides an overview of the client API. Section 74 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 75 describes the fundamentals of the Bigtable implementation, and Section 76 describes some of the refinements that we made to improve Bigtable's performance. Section 77 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 78, and discuss some lessons we learned in designing and supporting Bigtable in Section 79. Finally, Section 80 describes related work, and Section 81 presents our conclusions.

Section 82 describes the data model in more detail, and Section 83 provides an overview of the client API. Section 84 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 85 describes the fundamentals of the Bigtable implementation, and Section 86 describes some of the refinements that we made to improve Bigtable's performance. Section 87 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 88, and discuss some lessons we learned in designing and supporting Bigtable in Section 89. Finally, Section 90 describes related work, and Section 91 presents our conclusions.

Section 92 describes the data model in more detail, and Section 93 provides an overview of the client API. Section 94 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 95 describes the fundamentals of the Bigtable implementation, and Section 96 describes some of the refinements that we made to improve Bigtable's performance. Section 97 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 98, and discuss some lessons we learned in designing and supporting Bigtable in Section 99. Finally, Section 100 describes related work, and Section 101 presents our conclusions.

Section 102 describes the data model in more detail, and Section 103 provides an overview of the client API. Section 104 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 105 describes the fundamentals of the Bigtable implementation, and Section 106 describes some of the refinements that we made to improve Bigtable's performance. Section 107 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 108, and discuss some lessons we learned in designing and supporting Bigtable in Section 109. Finally, Section 110 describes related work, and Section 111 presents our conclusions.

Section 112 describes the data model in more detail, and Section 113 provides an overview of the client API. Section 114 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 115 describes the fundamentals of the Bigtable implementation, and Section 116 describes some of the refinements that we made to improve Bigtable's performance. Section 117 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 118, and discuss some lessons we learned in designing and supporting Bigtable in Section 119. Finally, Section 120 describes related work, and Section 121 presents our conclusions.

Section 122 describes the data model in more detail, and Section 123 provides an overview of the client API. Section 124 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 125 describes the fundamentals of the Bigtable implementation, and Section 126 describes some of the refinements that we made to improve Bigtable's performance. Section 127 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 128, and discuss some lessons we learned in designing and supporting Bigtable in Section 129. Finally, Section 130 describes related work, and Section 131 presents our conclusions.

Section 132 describes the data model in more detail, and Section 133 provides an overview of the client API. Section 134 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 135 describes the fundamentals of the Bigtable implementation, and Section 136 describes some of the refinements that we made to improve Bigtable's performance. Section 137 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 138, and discuss some lessons we learned in designing and supporting Bigtable in Section 139. Finally, Section 140 describes related work, and Section 141 presents our conclusions.

Section 142 describes the data model in more detail, and Section 143 provides an overview of the client API. Section 144 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 145 describes the fundamentals of the Bigtable implementation, and Section 146 describes some of the refinements that we made to improve Bigtable's performance. Section 147 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 148, and discuss some lessons we learned in designing and supporting Bigtable in Section 149. Finally, Section 150 describes related work, and Section 151 presents our conclusions.

Section 152 describes the data model in more detail, and Section 153 provides an overview of the client API. Section 154 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 155 describes the fundamentals of the Bigtable implementation, and Section 156 describes some of the refinements that we made to improve Bigtable's performance. Section 157 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 158, and discuss some lessons we learned in designing and supporting Bigtable in Section 159. Finally, Section 160 describes related work, and Section 161 presents our conclusions.

Section 162 describes the data model in more detail, and Section 163 provides an overview of the client API. Section 164 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 165 describes the fundamentals of the Bigtable implementation, and Section 166 describes some of the refinements that we made to improve Bigtable's performance. Section 167 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 168, and discuss some lessons we learned in designing and supporting Bigtable in Section 169. Finally, Section 170 describes related work, and Section 171 presents our conclusions.

Section 172 describes the data model in more detail, and Section 173 provides an overview of the client API. Section 174 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 175 describes the fundamentals of the Bigtable implementation, and Section 176 describes some of the refinements that we made to improve Bigtable's performance. Section 177 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 178, and discuss some lessons we learned in designing and supporting Bigtable in Section 179. Finally, Section 180 describes related work, and Section 181 presents our conclusions.

## MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

### Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key-value pair to generate a set of intermediate key-value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

### 1 Introduction

Over the past five years, the authors and many others at Google have implemented hundreds of special-purpose computations that process large amounts of raw data, such as crawled documents, web request logs, etc., to compute various kinds of derived data, such as inverted indices, various representations of the graph structure of web documents, summaries of the number of pages crawled per host, the set of most frequent queries in a

## Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

{fay,jeff,sanjay,wilson,dean,mike,tushar,fikes,gruber}@google.com

Google, Inc.

### Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this paper we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

### 1 Introduction

Over the last two and a half years we have designed, implemented, and deployed a distributed storage system for managing structured data at Google called Bigtable. Bigtable is designed to reliably scale to petabytes of data and thousands of machines. Bigtable has achieved several goals: wide applicability, scalability, high performance, and wide availability. Bigtable is used by more than sixty Google products and projects, including Google Analytics, Google Finance, Orkut, Personalized Search, Wirefly, and Google Earth. These products use Bigtable for a variety of demanding workloads, which range from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users.

The Bigtable clusters used by these products span a wide range of configurations, from a handful to thousands of servers, and store up to several hundred terabytes of data. In many ways, Bigtable resembles a database: it shares many implementation strategies with databases. Parallel databases [14] and main-memory databases [13] have

achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemes. Finally, Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

Section 2 describes the data model in more detail, and Section 3 provides an overview of the client API. Section 4 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 5 describes the fundamentals of the Bigtable implementation, and Section 6 describes some of the refinements that we made to improve Bigtable's performance. Section 7 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 8, and discuss some lessons we learned in designing and supporting Bigtable in Section 9. Finally, Section 10 describes related work, and Section 11 presents our conclusions.

### 2 Data Model

A Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

(row, string, column, string, timeInMS) → string

To appear in OSDI 2004

1

To appear in OSDI 2006

1

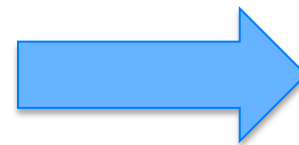
- The Google File System (2003)
- MapReduce: Simplified Data Processing... (2004)
- BigTable: A Distributed Storage System... (2006)



# Map-Reduce Example

- Input is file with one document per record
- User specifies map function
  - key = document URL
  - Value = terms that document contains

("doc cdickens",  
"it was the best of times")



map

"it", 1  
"was", 1  
"the", 1  
"best", 1

# Example (cont'd)

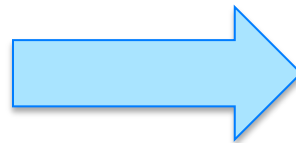
- MapReduce library gathers together all pairs with the same key value (shuffle/sort phase)
- The user-defined reduce function combines all the values associated with the same key

key = "it"  
values = 1, 1

key = "was"  
values = 1, 1

key = "best"  
values = 1

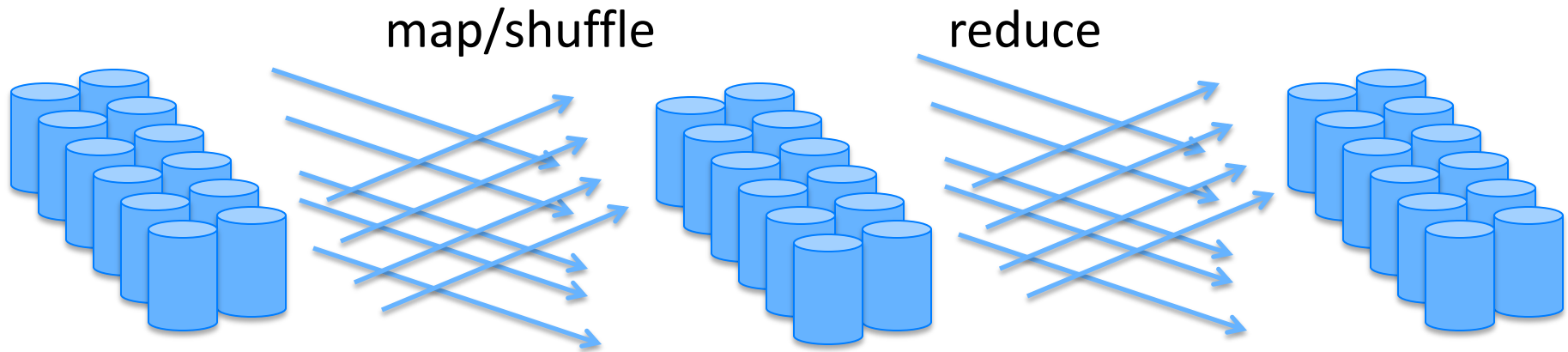
key = "worst"  
values = 1



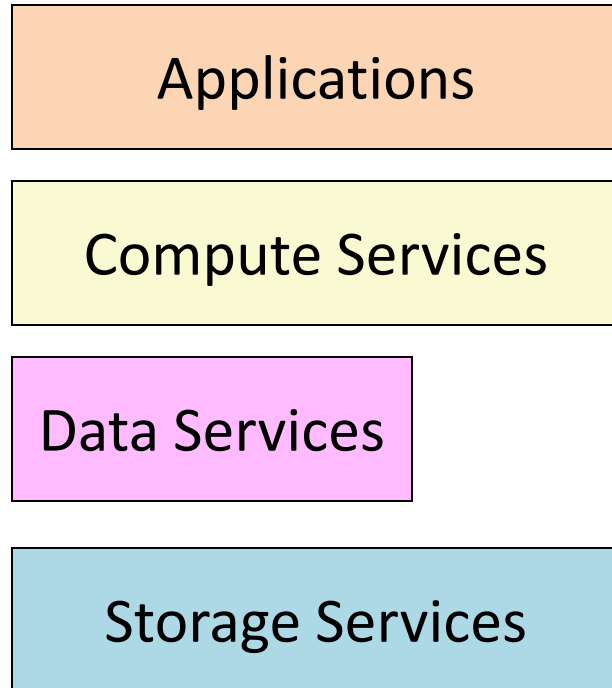
reduce

"it", 2  
"was", 2  
"best", 1  
"worst", 1

# Applying MapReduce to the Data in Storage Cloud



# Google's Large Data Cloud



Google's MapReduce

Google's BigTable

Google File System (GFS)

Google's Stack

# Hadoop's Large Data Cloud

Applications

Compute Services

Data Services

Storage Services

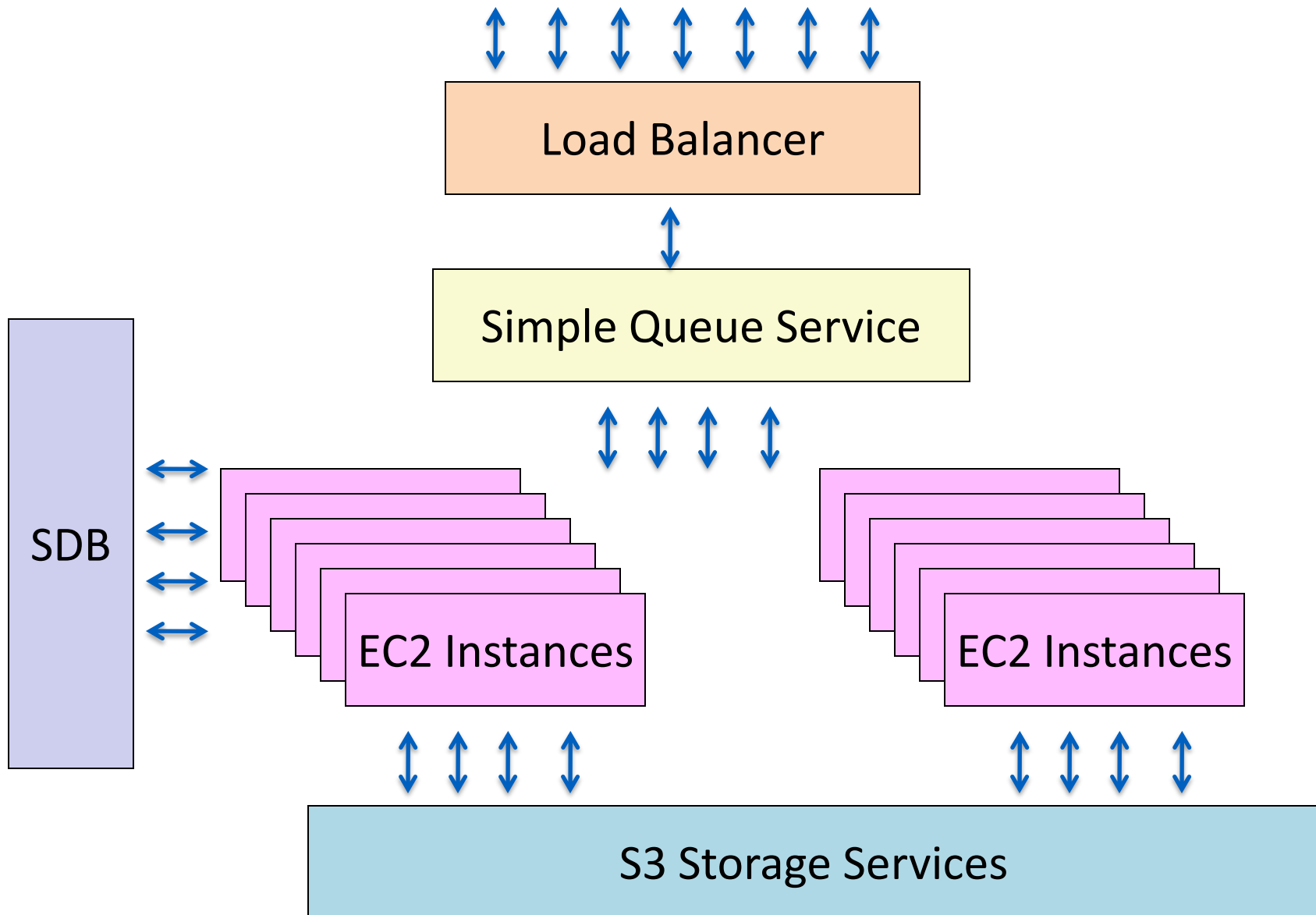
Hadoop's Stack

Hadoop's MapReduce

NoSQL Databases

Hadoop Distributed File  
System (HDFS)

# Amazon Style Data Cloud



# Evolution of NoSQL Databases

- Standard architecture for simple web apps:
  - Front end load balanced web servers
  - Business logic layer in the middle
  - Backend database
- Databases do not scale well with very large numbers of users or very large amounts of data
- Alternatives include
  - Sharded (partitioned) databases
  - master-slave databases
  - memcached

# NoSQL Systems

- Suggests No SQL support, also Not Only SQL
- One or more of the ACID properties not supported
- Joins generally not supported
- Usually flexible schemas
- Some well known examples: Google's BigTable, Amazon's S3 & Facebook's Cassandra
- Several recent open source systems



# Different Types of NoSQL Systems

- Distributed Key-Value Systems
  - Amazon's S3 Key-Value Store (Dynamo)
  - Voldemort
- Column-based Systems
  - BigTable
  - HBase
  - Cassandra
- Document-based systems
  - CouchDB

# Cassandra vs MySQL Comparison

- MySQL > 50 GB Data  
Writes Average : ~300 ms  
Reads Average : ~350 ms
- Cassandra > 50 GB Data  
Writes Average : 0.12 ms  
Reads Average : 15 ms

Source: Avinash Lakshman, Prashant Malik, Cassandra  
Structured Storage System over a P2P Network, [static.last.fm/johan/nosql-20090611/cassandra\\_nosql.pdf](http://static.last.fm/johan/nosql-20090611/cassandra_nosql.pdf)

# CAP Theorem

- Proposed by Eric Brewer, 2000
- Three properties of a system: consistency, availability and partitions
- You can have at most two of these three properties for any shared-data system
- Scale out requires partitions
- Most large web-based systems choose availability over consistency

# Eventual Consistency

- All updates eventually propagate through the system and all nodes will eventually be consistent (assuming no more updates)
- Eventually, a node is either updated or removed from service.
- Can be implemented with Gossip protocol
- Amazon's Dynamo popularized this approach
- Sometimes this is called BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency), as opposed to ACID

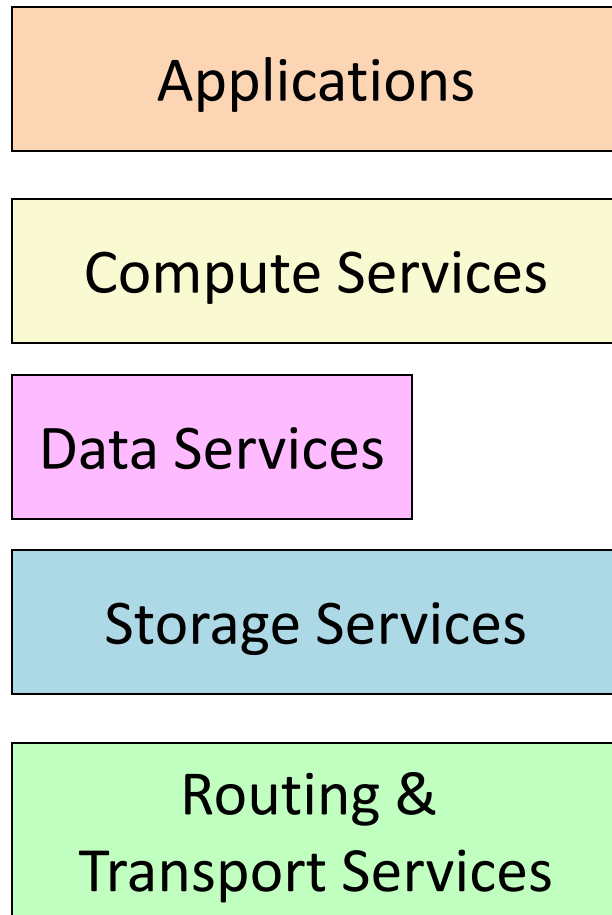
# Part 5. Sector Architecture



# Design Objectives

1. Provide Internet scale data storage for large data
  - Support multiple data centers connected by high speed wide networks
2. Simplify data intensive computing for a larger class of problems than covered by MapReduce
  - Support applying User Defined Functions to the data managed by a storage cloud, with transparent load balancing and fault tolerance

# Sector's Large Data Cloud



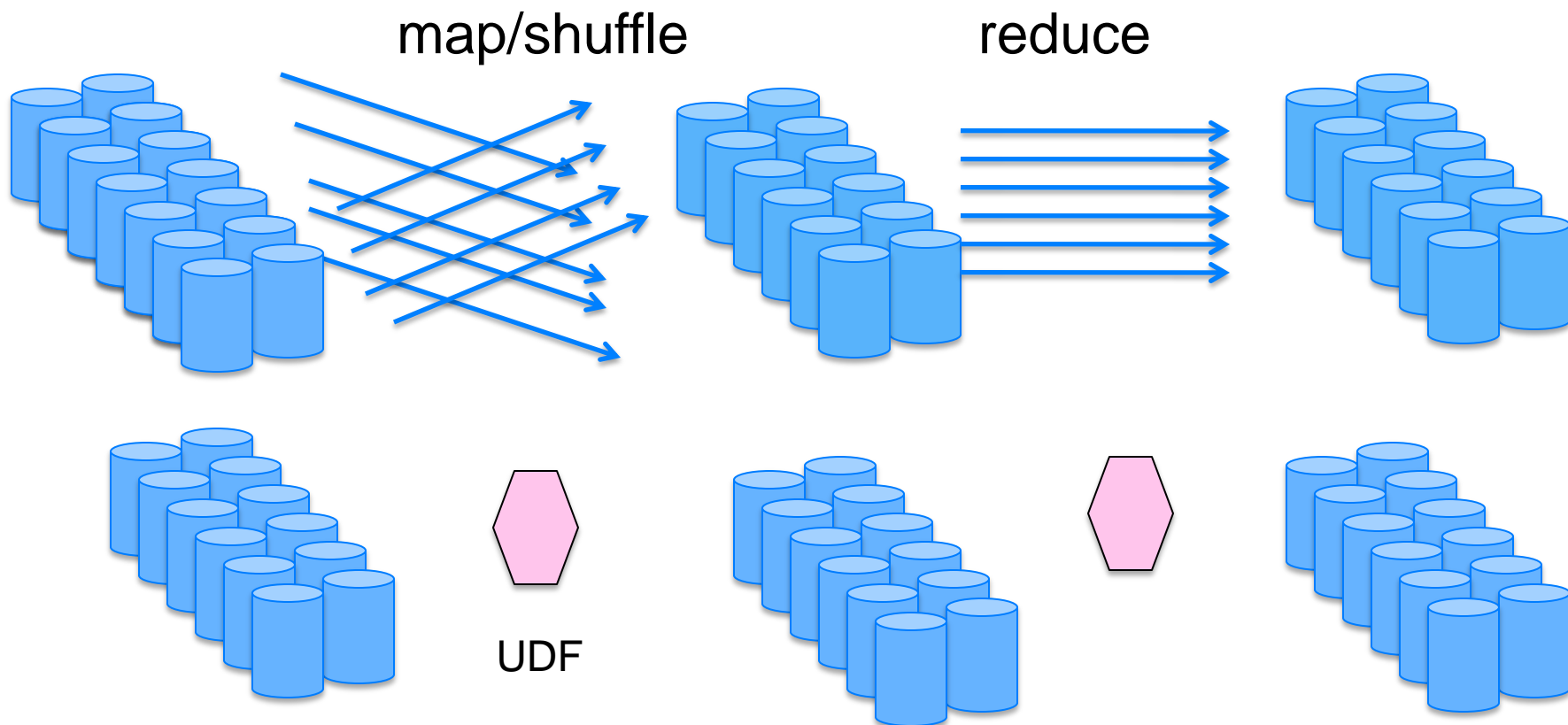
Sector's Stack

Sphere's UDFs

Sector's Distributed File  
System (SDFS)

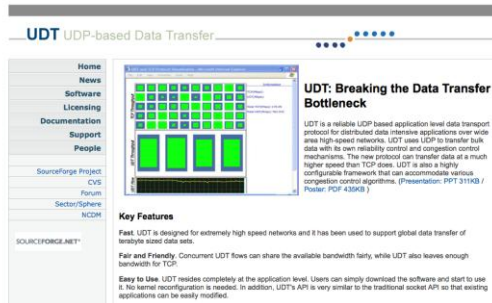
UDP-based Data Transport  
Protocol (UDT)

# Apply User Defined Functions (UDF) to Files in Storage Cloud

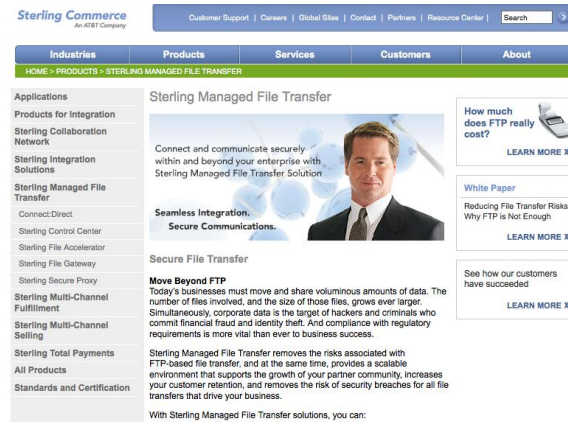




# UDT



udt.sourceforge.net



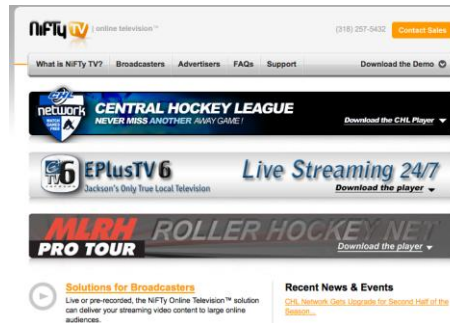
Sterling Commerce



Movie2Me



Globus



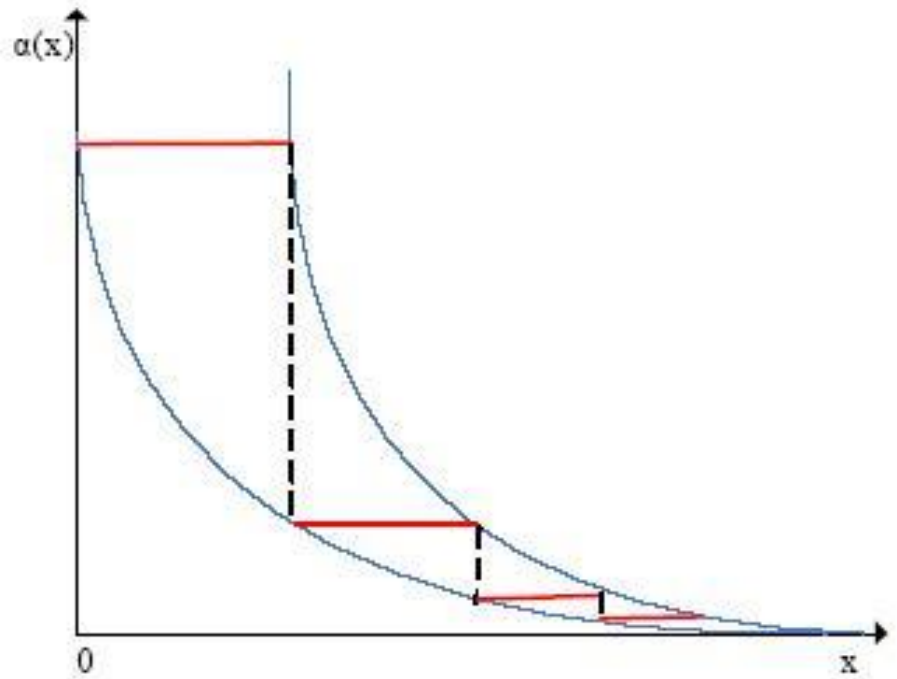
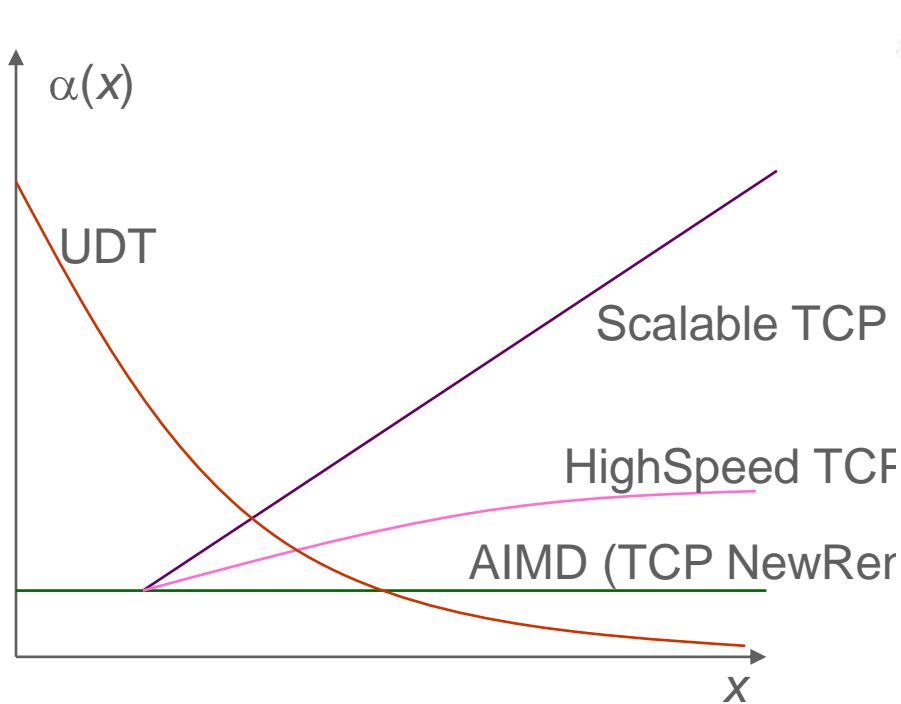
Nifty TV



Power Folder

UDT has been downloaded 25,000+ times

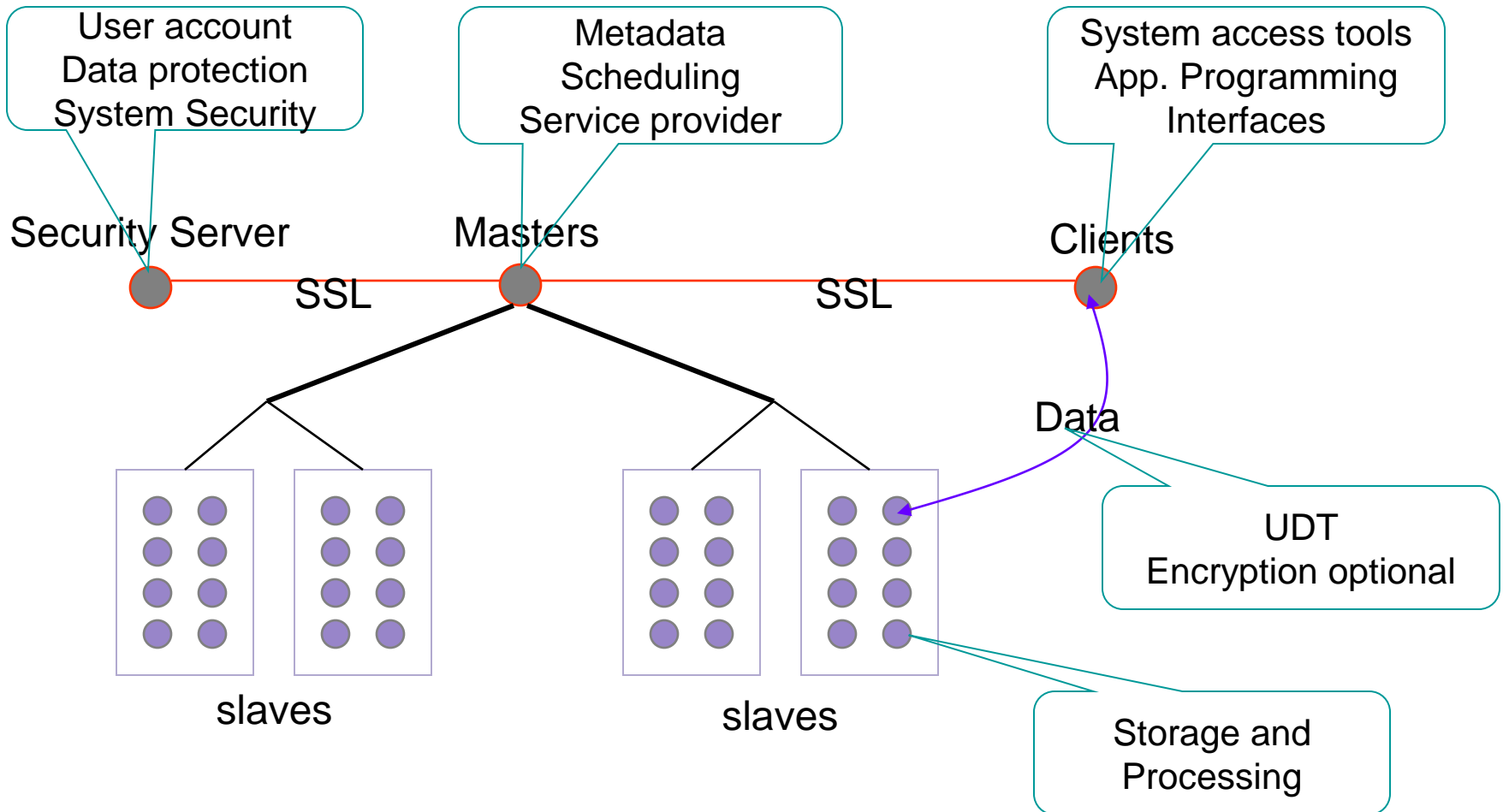
# Alternatives to TCP – Decreasing Increases AIMD Protocols



$x \leftarrow x + \alpha(x)$       increase of packet sending rate  $x$

$x \leftarrow (1 - \beta) x$       decrease factor

# System Architecture



	<b>Hadoop DFS</b>	<b>Sector DFS</b>
Storage Cloud	Block-based file system	File-based
Programming Model	MapReduce	UDF & MapReduce
Protocol	TCP	UDP-based protocol (UDT)
Replication	At write	At write or period.
Security	Not yet	HIPAA capable
Language	Java	C++

	<b>MapReduce</b>	<b>Sphere</b>
Storage	Disk data	Disk & in-memory
Processing	Map followed by Reduce	Arbitrary user defined functions
Data exchanging	Reducers pull results from mappers	UDF's push results to bucket files
Input data locality	Input data is assigned to nearest mapper	Input data is assigned to nearest UDF
Output data locality	NA	Can be specified

# Terasort Benchmark

	1 Rack	2 Racks	3 Racks	4 Racks
Nodes	32	64	96	128
Cores	128	256	384	512
Hadoop	85m 49s	37m 0s	25m 14s	17m 45s
Sector	28m 25s	15m 20s	10m 19s	7m 56s
Speed up	3.0	2.4	2.4	2.2

Sector/Sphere 1.24a, Hadoop 0.20.1 with no replication on  
Phase 2 of Open Cloud Testbed with co-located racks.

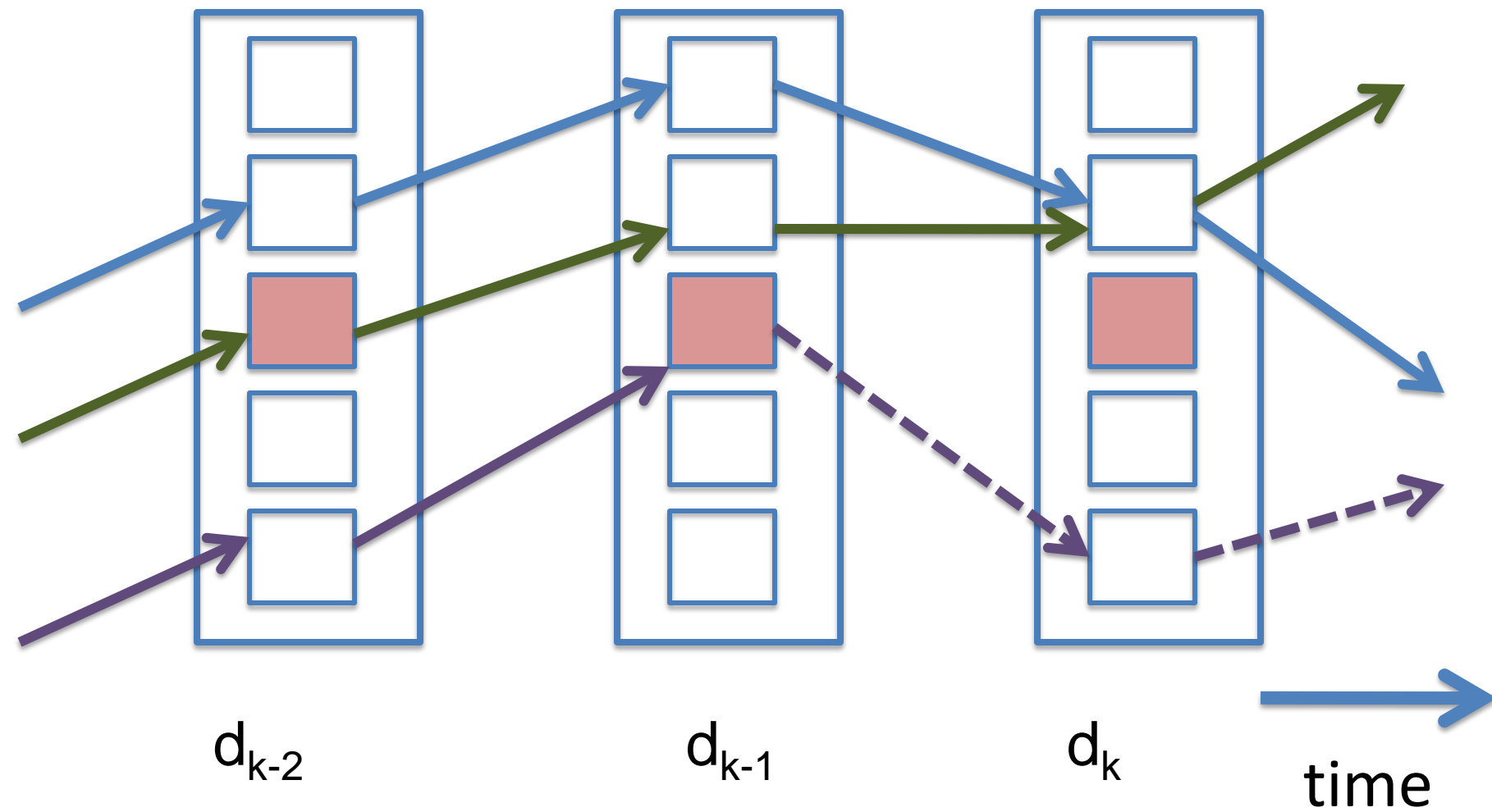
# MalStone



sites



entities



# MalStone Benchmark

	MalStone A	MalStone B
Hadoop	455m 13s	840m 50s
Hadoop streaming with Python	87m 29s	142m 32s
Sector/Sphere	33m 40s	43m 44s
Speed up (Sector v Hadoop)	13.5x	19.2x

Sector/Sphere 1.20, Hadoop 0.18.3 with no replication on Phase 1 of Open Cloud Testbed in a single rack. Data consisted of 20 nodes with 500 million 100-byte records / node.



Disks

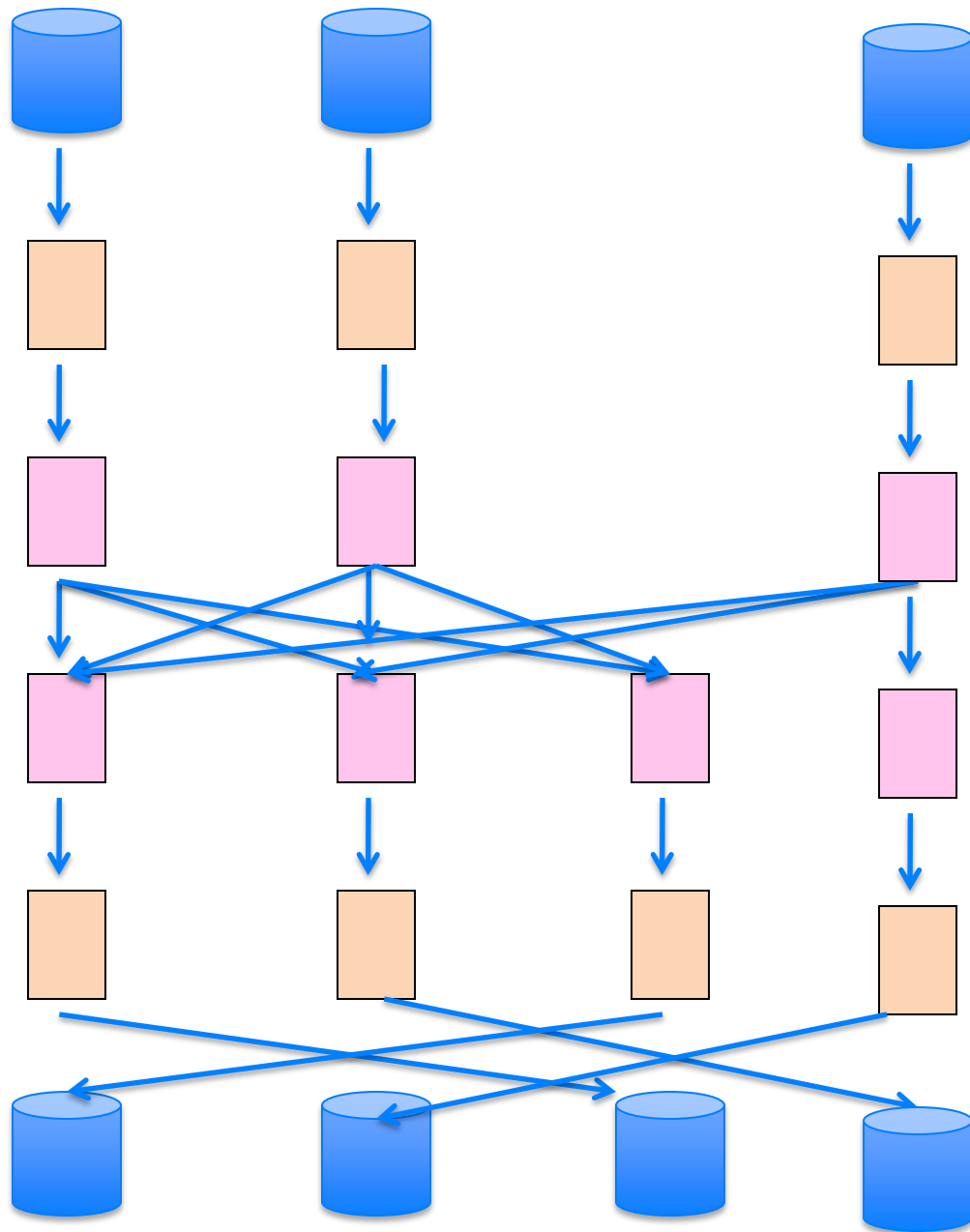
Input  
Segments

UDF

Bucket  
Writers

Output  
Segments

Disks



- Files not split into blocks
- Directory directives
- In-memory objects

# Sector Summary

- Sector is fastest open source large data cloud
  - As measured by MalStone & Terasort
- Sector is easy to program
  - UDFs, MapReduce & Python over streams
- Sector does not require extensive tuning
- Sector is secure
  - A HIPAA compliant Sector cloud is being launched
- Sector is reliable
  - Sector supports multiple active master node servers

## SDSS Data Distribution using Sector and UDT

search

go

Overview

Download Instructions

Software

Nodes Status

Downloading Records

Documentation

Technical Contact

Related Links

NOES

UDT

SciApp

Teraflow Testbed

### Overview

Using this web site, you can download the Sloan Digital Sky Survey (SDSS) data if you have access to a high speed wide area network. For example, if your organization is attached to the National Lambda Rail or Internet2's Abilene Network, then you should be able to download the entire SDSS BESTDR5 catalog data set in less than five hours.

In general, it can be quite challenging to use effectively the available bandwidth over a wide area, high performance network. This project uses the UDP-based Data Transfer Protocol or UDT, which has been developed by the National Center for Data Mining (NCDM) at the University of Illinois at Chicago to make effective use of the bandwidth available from high performance wide area networks.

The project is supported by the National Science Foundation through the grant SCI II: The TeraFlow Project: High Performance Flows for Mining Large Distributed Data Archives, Award SCI-0430781.

### Sloan Digital Sky Survey (SDSS)



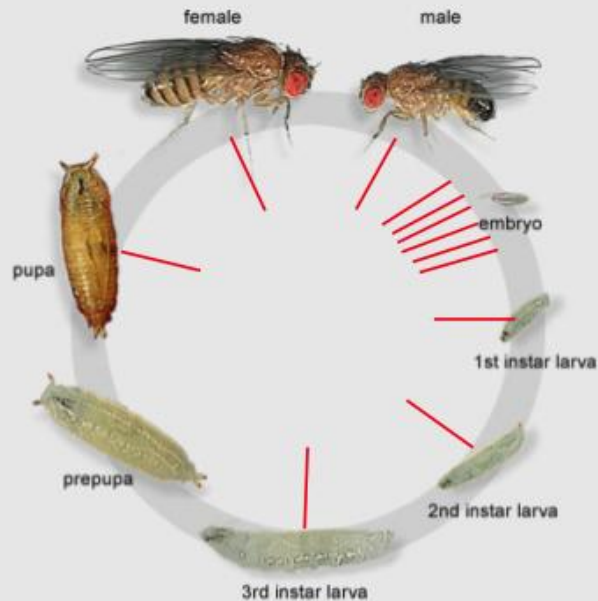
The SDSS is systematically mapping a quarter of the entire sky, producing a detailed image of it, and determining the positions and absolute brightness of more than 100 million celestial objects. It is also measuring the distances to a million of the nearest galaxies, giving us a three-dimensional picture of the universe through a volume one hundred times larger than that explored to date. SDSS is also recording the distances to 100,000 quasars — the most distant objects known — giving us unprecedented knowledge of the distribution of matter to the edge of the visible universe.

The SDSS completed its first phase of operations — SDSS-I — in June, 2005. Over the course of five years, SDSS-I imaged more than 8,000 square degrees of the sky in five band passes, detecting nearly 200 million celestial objects, and it measured spectra of more than 675,000 galaxies, 90,000 quasars, and 185,000 stars. These data have supported studies ranging from asteroids and nearby stars to the large scale structure of the Universe.

The most recent data product is DR6, which was released on June, 2007.

# Part 6. Sector Applications

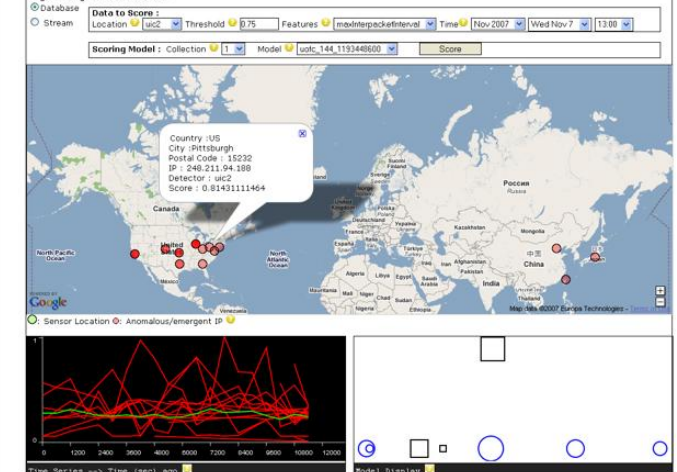
## The life cycle of *Drosophila melanogaster*



### Experiment Units

E-4-8h	E-8-12h	E-12-16h	E-16-20h	E-20-24h	L1	L2	L3	Pupae	AdultFemale	AdultMale
<a href="#">CD 385</a>	<a href="#">CD 393</a>	<a href="#">CD 399</a>	<a href="#">CD 407</a>	<a href="#">CD 415</a>	<a href="#">CD 423</a>	<a href="#">CD 431</a>	<a href="#">CD 438</a>	<a href="#">CD 446</a>	<a href="#">CD 454</a>	<a href="#">CD 462</a>
<a href="#">CD 386</a>	<a href="#">CD 394</a>	<a href="#">CD 400</a>	<a href="#">CD 408</a>	<a href="#">CD 416</a>	<a href="#">CD 424</a>	<a href="#">CD 432</a>	<a href="#">CD 439</a>	<a href="#">CD 447</a>	<a href="#">CD 455</a>	<a href="#">CD 463</a>
<a href="#">CD 387</a>	<a href="#">CD 395</a>	<a href="#">CD 401</a>	<a href="#">CD 409</a>	<a href="#">CD 417</a>	<a href="#">CD 425</a>	<a href="#">CD 433</a>	<a href="#">CD 440</a>	<a href="#">CD 448</a>	<a href="#">CD 456</a>	<a href="#">CD 464</a>
<a href="#">CD 388</a>	N/A	<a href="#">CD 402</a>	<a href="#">CD 410</a>	<a href="#">CD 418</a>	<a href="#">CD 426</a>	<a href="#">CD 434</a>	<a href="#">CD 441</a>	<a href="#">CD 449</a>	<a href="#">CD 457</a>	<a href="#">CD 465</a>
<a href="#">CD 389</a>	N/A	<a href="#">CD 403</a>	<a href="#">CD 411</a>	<a href="#">CD 419</a>	<a href="#">CD 427</a>	<a href="#">CD 435</a>	<a href="#">CD 442</a>	<a href="#">CD 450</a>	<a href="#">CD 458</a>	<a href="#">CD 466</a>
<a href="#">CD 390</a>	<a href="#">CD 396</a>	<a href="#">CD 404</a>	<a href="#">CD 412</a>	<a href="#">CD 420</a>	<a href="#">CD 428</a>	<a href="#">CD 436</a>	<a href="#">CD 443</a>	<a href="#">CD 451</a>	<a href="#">CD 459</a>	<a href="#">CD 467</a>
<a href="#">CD 391</a>	<a href="#">CD 397</a>	<a href="#">CD 405</a>	<a href="#">CD 413</a>	<a href="#">CD 421</a>	<a href="#">CD 429</a>	<a href="#">CD 437</a>	<a href="#">CD 444</a>	<a href="#">CD 452</a>	<a href="#">CD 460</a>	<a href="#">CD 468</a>
<a href="#">CD 392</a>	<a href="#">CD 398</a>	<a href="#">CD 406</a>	<a href="#">CD 414</a>	<a href="#">CD 422</a>	<a href="#">CD 430</a>	<a href="#">CD 438</a>	<a href="#">CD 445</a>	<a href="#">CD 453</a>	<a href="#">CD 461</a>	<a href="#">CD 469</a>


### Angle Emergent Network Demo



# App 1: Bionimbus

[Home](#) | [About Us](#) | [Contact Us](#)

# Cistrack



## Public Cistrack Data

[\[Drosophila Chromatin Time Course\]](#) - modENCODE

[\[Drosophila Insulator\]](#) - modENCODE

[\[All Data\]](#) - modENCODE

[Browse Flynet](#)

Browse and download public Cistrack data

- [\[By Experiment\]](#)
- [\[By File\]](#)
- [\[By Experimental Unit\]](#)

## Lookup Cistrack accession number

## Welcome to Cistrack

Cistrack supports data distribution for the Drosophila modENCODE cis-regulatory project (NHGRI contract U01HG004264) and the Chicago Center for Systems Biology (NIGMS grant P50GN081892).

Some browsers experience a problem when downloading public data. If you are asked to login when accessing public data, please close the web page and try again. This problem will be fixed with the next release.

## Cistrack Users

[Login to Cistrack](#) or [\[Register\]](#)

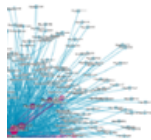
[Upload data files](#)

[Annotate uploaded files with metadata](#)

[Cistrack Wiki](#)

www.bionimbus.org

# App 2. Sector Application: Cistrack & Flynet



flynet version 1.0

[Home](#) | [Search](#) | [Advanced Search](#) | [Resource](#) | [Help](#) | [Contact](#)

Available Tracks:  
(Drag → to view)

Motif

CDS

Exons

Cytological band

Non coding RNA

Natural transposon

Transgene  
insertion site

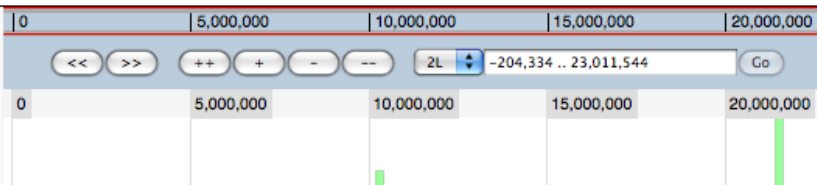
Oligonucleotides

enhancer

Mutation: point  
mutation

Mutation:  
sequence variant

Mutation:  
aberration junction



Cistrack

Drosophila Chromatin Time-Course

This is a dataset generated by the Drosophila Regulatory Elements modENCODE Project led by Kevin P. White at the University of Chicago. It contains ChIP-chip data on Agilent 244K dual-color arrays for 6 Histone modifications (H3K9me3, H3K27me3, H3K4me3, H3K4me1, H3K27Ac, H3K9Ac), PolII and CBP/p300. Each factor has been studied for 12 different time-points of Drosophila development.

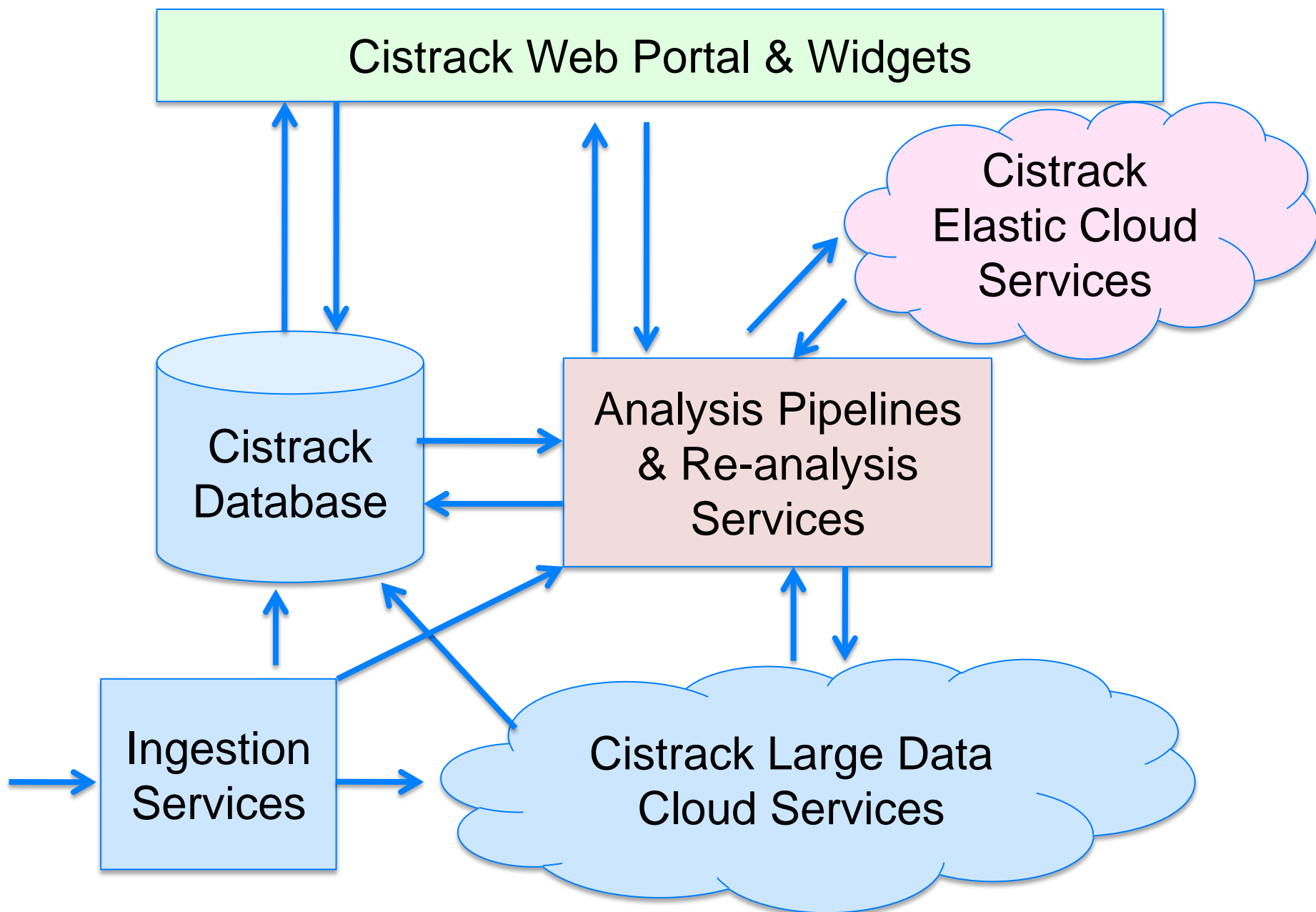
	E-0-4h	E-4-8h	E-8-12h	E-12-16h	E-16-20h	E-20-24h	L1	L2	L3	Pupae	AdultFemale	AdultMale
H3K4Me3	<a href="#">CD_4</a>	<a href="#">CD_11</a>	<a href="#">N/A</a>	<a href="#">CD_17</a>	<a href="#">CD_23</a>	<a href="#">CD_30</a>	<a href="#">CD_37</a>	<a href="#">CD_44</a>	<a href="#">CD_51</a>	<a href="#">CD_58</a>	<a href="#">CD_72</a>	<a href="#">CD_65</a>
H3K4Me1	<a href="#">CD_3</a>	<a href="#">CD_10</a>	<a href="#">N/A</a>	<a href="#">CD_16</a>	<a href="#">CD_22</a>	<a href="#">CD_29</a>	<a href="#">CD_36</a>	<a href="#">CD_43</a>	<a href="#">CD_50</a>	<a href="#">CD_57</a>	<a href="#">CD_71</a>	<a href="#">CD_64</a>
H3K9Ac	<a href="#">CD_5</a>	<a href="#">CD_12</a>	<a href="#">N/A</a>	<a href="#">CD_18</a>	<a href="#">CD_24</a>	<a href="#">CD_31</a>	<a href="#">CD_38</a>	<a href="#">CD_45</a>	<a href="#">CD_52</a>	<a href="#">CD_59</a>	<a href="#">CD_73</a>	<a href="#">CD_66</a>
H3K9Me3	<a href="#">CD_6</a>	<a href="#">CD_13</a>	<a href="#">N/A</a>	<a href="#">CD_19</a>	<a href="#">CD_25</a>	<a href="#">CD_32</a>	<a href="#">CD_39</a>	<a href="#">CD_46</a>	<a href="#">CD_53</a>	<a href="#">CD_60</a>	<a href="#">CD_74</a>	<a href="#">CD_67</a>
H3K27Ac	<a href="#">CD_7</a>	<a href="#">CD_14</a>	<a href="#">N/A</a>	<a href="#">CD_20</a>	<a href="#">CD_26</a>	<a href="#">CD_33</a>	<a href="#">CD_40</a>	<a href="#">CD_47</a>	<a href="#">CD_54</a>	<a href="#">CD_61</a>	<a href="#">CD_75</a>	<a href="#">CD_68</a>
H3K27Me3	<a href="#">CD_8</a>	<a href="#">CD_15</a>	<a href="#">N/A</a>	<a href="#">CD_21</a>	<a href="#">CD_27</a>	<a href="#">CD_34</a>	<a href="#">CD_41</a>	<a href="#">CD_48</a>	<a href="#">CD_55</a>	<a href="#">CD_62</a>	<a href="#">CD_76</a>	<a href="#">CD_69</a>
PolII	<a href="#">CD_9</a>	<a href="#">N/A</a>	<a href="#">N/A</a>	<a href="#">N/A</a>	<a href="#">CD_28</a>	<a href="#">CD_35</a>	<a href="#">CD_42</a>	<a href="#">CD_49</a>	<a href="#">CD_56</a>	<a href="#">CD_63</a>	<a href="#">CD_77</a>	<a href="#">CD_70</a>
CBP	<a href="#">N/A</a>	<a href="#">N/A</a>	<a href="#">N/A</a>	<a href="#">N/A</a>	<a href="#">N/A</a>	<a href="#">N/A</a>	<a href="#">N/A</a>	<a href="#">N/A</a>	<a href="#">N/A</a>	<a href="#">N/A</a>	<a href="#">N/A</a>	<a href="#">N/A</a>

[\[logout\]](#)

[About Cistrack](#)

Cistrack is powered by CUBioS

[Return home](#)



# App 3: Bulk Download of the SDSS

Source	Destin.	LLPR*	Link	Bandwidth
Chicago	Greenbelt	0.98	1 Gb/s	615 Mb/s
Chicago	Austin	0.83	10 Gb/s	8000 Mb/s

## SDSS Data Distribution using Sector and UDT

search

go

Overview

Download Instructions

Software

Nodes Status

Downloading Records

Documentation

Technical Contact

Related Links

NCDM

SDSS

UDT

Sector

Teraflow Testbed


Overview

Using this web site, you can download the Sloan Digital Sky Survey (SDSS) data if you have access to a high speed wide area network. For example, if your organization is attached to the [National Lambda Rail](#) or Internet2's [Abilene Network](#), then you should be able to download the entire SDSS BESTDR5 catalog data set in less than five hours.

In general, it can be quite challenging to use effectively the available bandwidth over a wide area, high performance network. This project uses the UDP-based Data Transfer Protocol or [UDT](#), which has been developed by the National Center for Data Mining (NCDM) at the University of Illinois at Chicago to make effective use of the bandwidth available from high performance wide area networks.

The project is supported by the National Science Foundation through the grant SCI II: The TeraFlow Project: High Performance Flows for Mining Large Distributed Data Archives, Award SCI-0430781.

Sloan Digital Sky Survey (SDSS)



The SDSS is systematically mapping a quarter of the entire sky, producing a detailed image of it, and determining the positions and absolute brightness of more than 100 million celestial objects. It is also measuring the distances to a million of the nearest galaxies, giving us a three-dimensional picture of the universe through a volume one hundred times larger than that explored to date. SDSS is also recording the distances to 100,000 quasars — the most distant objects known — giving us unprecedented knowledge of the distribution of matter to the edge of the visible universe.

The SDSS completed its first phase of operations — SDSS-I — in June, 2005. Over the course of five years, SDSS-I imaged more than 8,000 square degrees of the sky in five band passes, detecting nearly 200 million celestial objects, and it measured spectra of more than 675,000 galaxies, 90,000 quasars, and 185,000 stars. These data have supported studies ranging from asteroids and nearby stars to the large scale structure of the Universe.

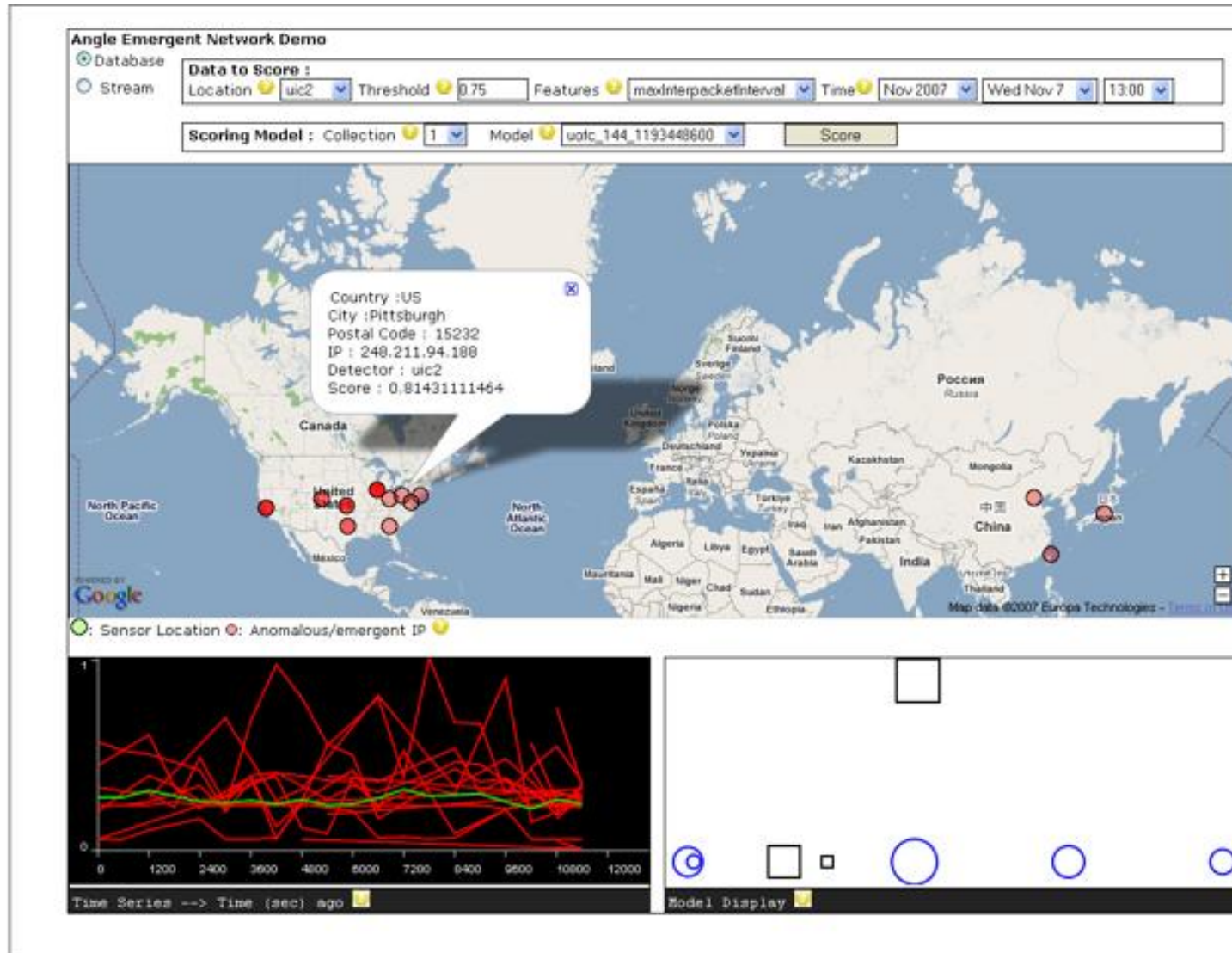
The most recent data product is DR6, which was released on June, 2007.

- LLPR = local / long distance performance
- Sector LLPR varies between 0.61 and 0.98

Recent Sloan Digital Sky Survey (SDSS) data release is 14 TB in size<sup>55</sup>



# App 4: Anomalies in Network Data






# Sector Applications

- Distributing the 15 TB Sloan Digital Sky Survey to astronomers around the world (with JHU, 2005)
- Managing and analyzing high throughput sequence data (Cistrack, University of Chicago, 2007).
- Detecting emergent behavior in distributed network data (Angle, won SC 07 Analytics Challenge)
- Wide area clouds (won SC 09 BWC with 100 Gbps wide area computation)
- New ensemble-based algorithms for trees
- Graph processing
- Image processing (OCC Project Matsu)

# Credits

**VeryCloud** accelerate your data

[company](#)  
[newsroom](#)  
[products](#)  
[services](#)  
[solutions](#)  
[resources](#)  
[contact us](#)  
  
[UDT](#)  
[Sector/Sphere](#)  
  
[Our Blog](#)



## High Performance Data Transfer, Storage, and Processing

VeryCloud provides solutions for high speed data transfer, high performance distributed data storage, and massive parallel data processing. With award-winning open source software and cutting-edge technologies on high speed networking and distributed computing, we can help significantly improve the performance of your systems that handle large data and reduce the software/hardware cost and development time.

- Sector was developed by Yunhong Gu from the University of Illinois at Chicago and [verycloud.com](http://verycloud.com)

# For More Information

For more information, please visit

[sector.sourceforge.net](http://sector.sourceforge.net)

[rgrossman.com](http://rgrossman.com) (Robert Grossman)

[users.lac.uic.edu/~yunhong](http://users.lac.uic.edu/~yunhong) (Yunhong Gu)