Running Map-Reduce Under Condor

Condor Project
Computer Sciences Department
University of Wisconsin-Madison

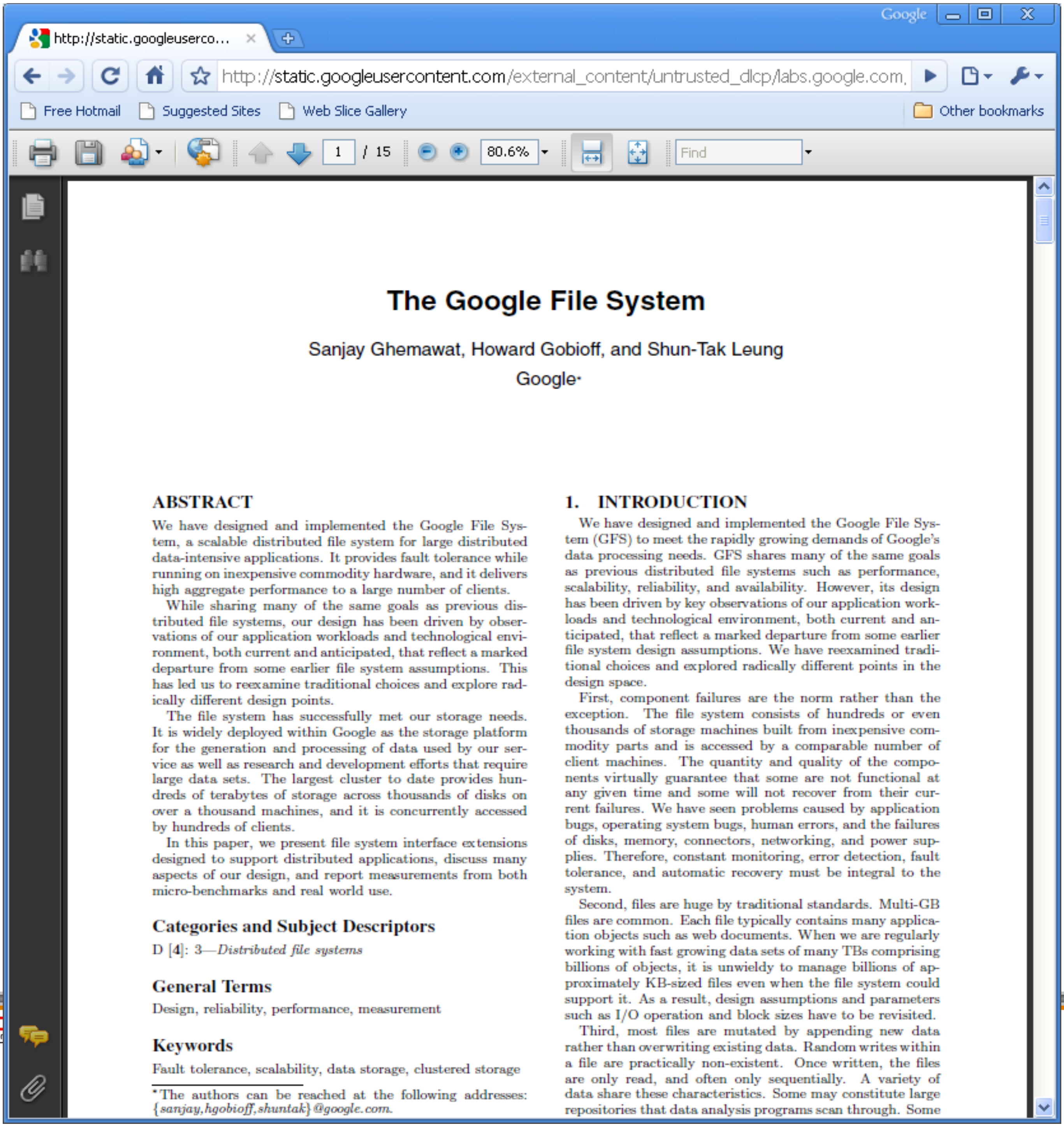
Cast of thousands

- Mihai Pop
- Michael Schatz
- Dan Sommer
  - University of Maryland Center for Computational Biology
- Faisal Khan, Ken Hahn UW
- David Schwartz, LMCG

In 2003…

http://labs.google.com/papers/gfs.html

http://labs.google.com/papers/mapreduce.html


http://static.googleuserco...
http://static.googleusercontent.com/external_content/untrusted_dlcp/labs.google.com,
Free Hotmail
Suggested Sites
Web Slice Gallery
Other bookmarks
1 / 15
80.6%
Find

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

ABSTRACT

We have designed and implemented the Google File System, 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 environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require large data sets. The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients.

In this paper, we present file system interface extensions designed to support distributed applications, discuss many aspects of our design, and report measurements from both micro-benchmarks and real world use.

Categories and Subject Descriptors

D [4]: 3—*Distributed file systems*

General Terms

Design, reliability, performance, measurement

Keywords

Fault tolerance, scalability, data storage, clustered storage

*The authors can be reached at the following addresses: *{sanjay,hgobioff,shuntak}@google.com.*

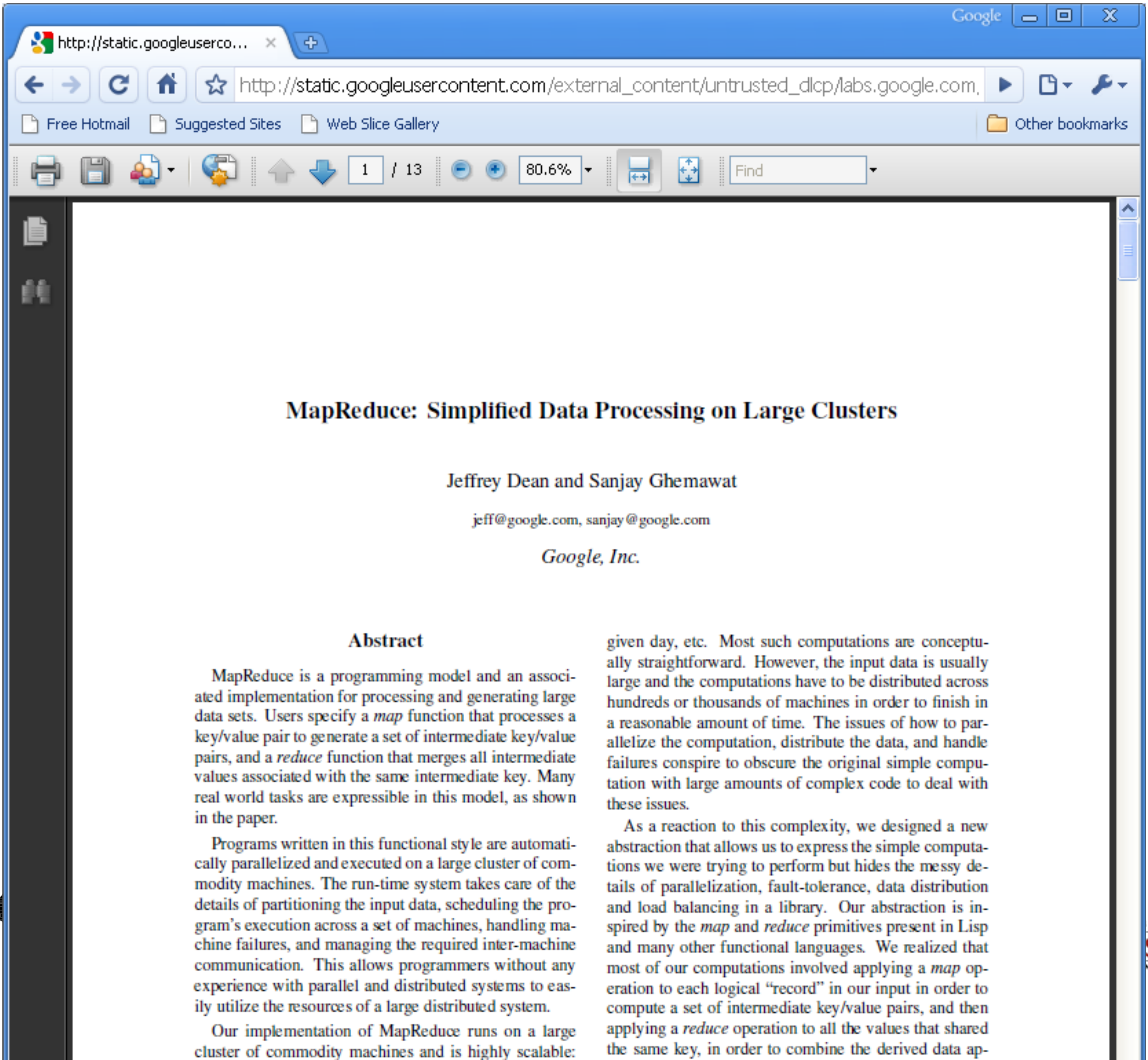
1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. 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 comparable number of client machines. The quantity and quality of the components virtually 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, human errors, and the failures of disks, memory, connectors, networking, and power supplies. Therefore, constant monitoring, error detection, fault tolerance, and automatic recovery must be integral to the system.

Second, files are huge 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 many TBs comprising billions of objects, it is unwieldy to manage billions of approximately KB-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 revisited.

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 files are only read, and often only sequentially. A variety of data share these characteristics. Some may constitute large repositories that data analysis programs scan through. Some


http://static.googleuserco...
http://static.googleusercontent.com/external_content/untrusted_dlcp/labs.google.com,
Free Hotmail
Suggested Sites
Web Slice Gallery
Other bookmarks
1 / 13
80.6%
Find

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:

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 ap-

Shortly thereafter…

hadoop

Two main Hadoop parts

hadoop HDFS

Java

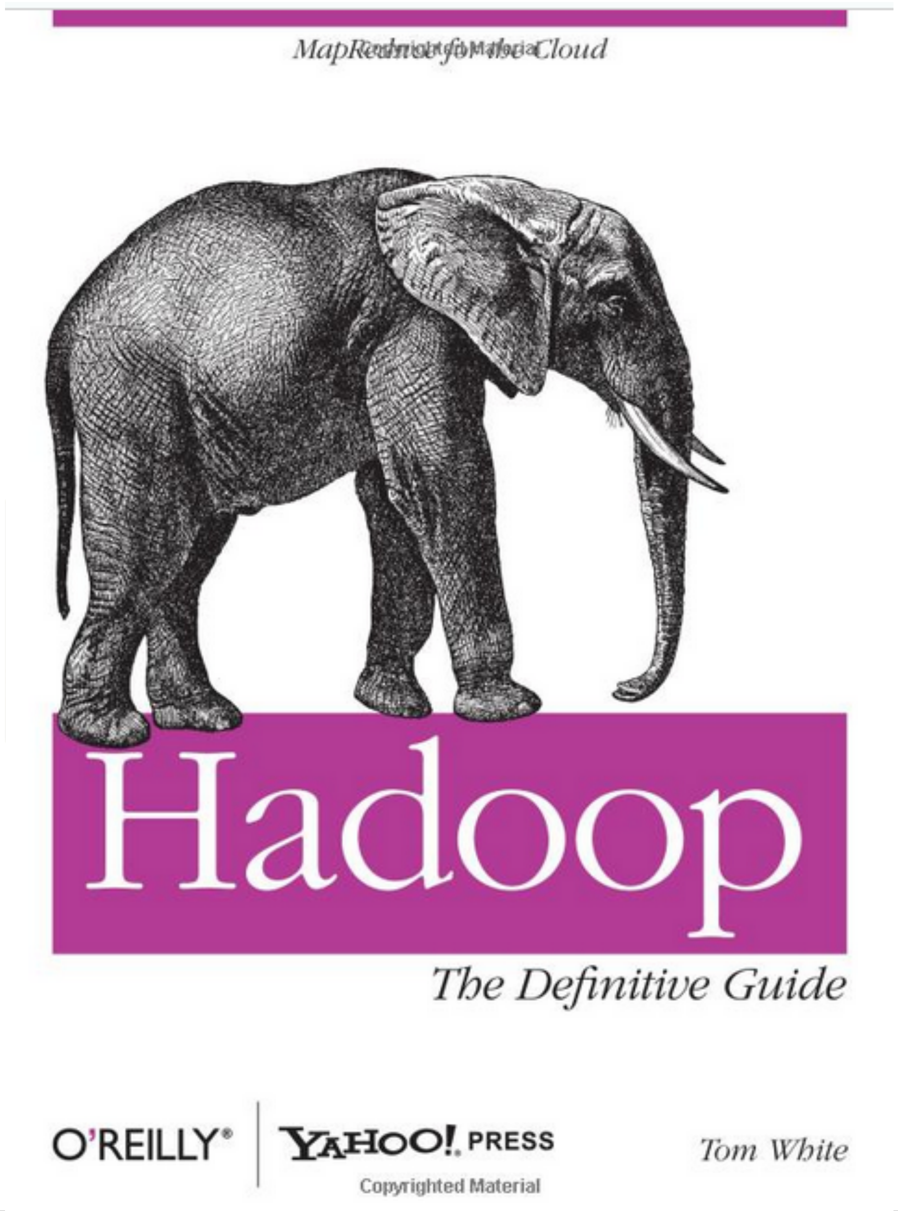
hadoop MapReduce

For more detail

CondorWeek 2009 talk

Dhruba Borthakur

http://www.cs.wisc.edu/condor/CondorWeek2009/condor_presentations/borthakur-hadoop_univ_research.ppt


MapReduce for the Cloud

Copyrighted Material

Hadoop

The Definitive Guide

O'REILLY® | YAHOO! PRESS

Tom White

Copyrighted Material

HDFS overview

- Making POSIX **distributed** file system go fast is easy…

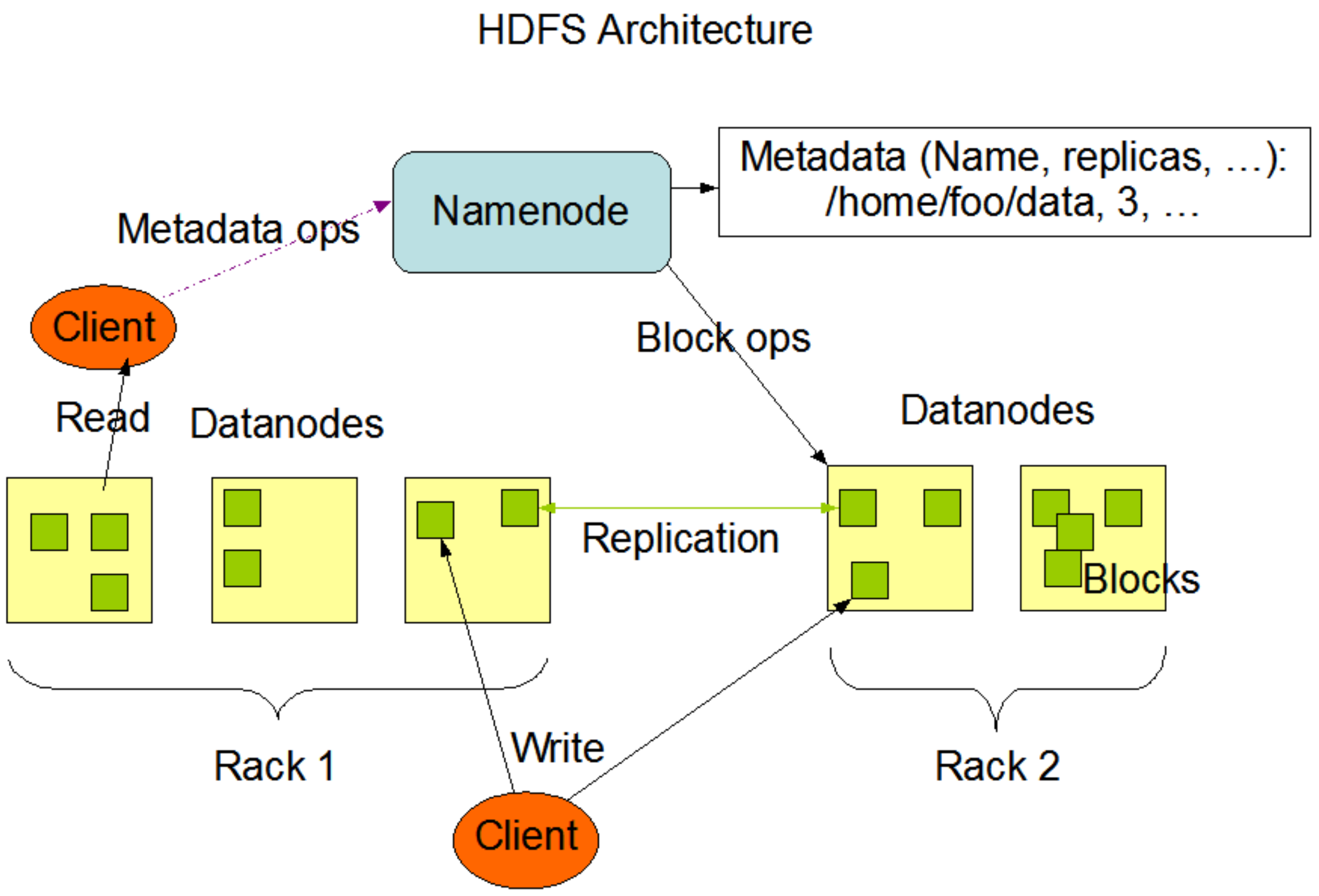
HDFS overview

- …If you get rid of the POSIX part
- Remove
  - Random access
  - Support for small files
  - authentication
  - In-kernel support

HDFS Overview

- Add in
  - Data replication
    - (key for distributed systems)
  - Command line utilities

HDFS Architecture



HDFS Condor Integration

- HDFS Daemons run under master
  - Management/control
- Added HAD support for namenode
- Added host based security

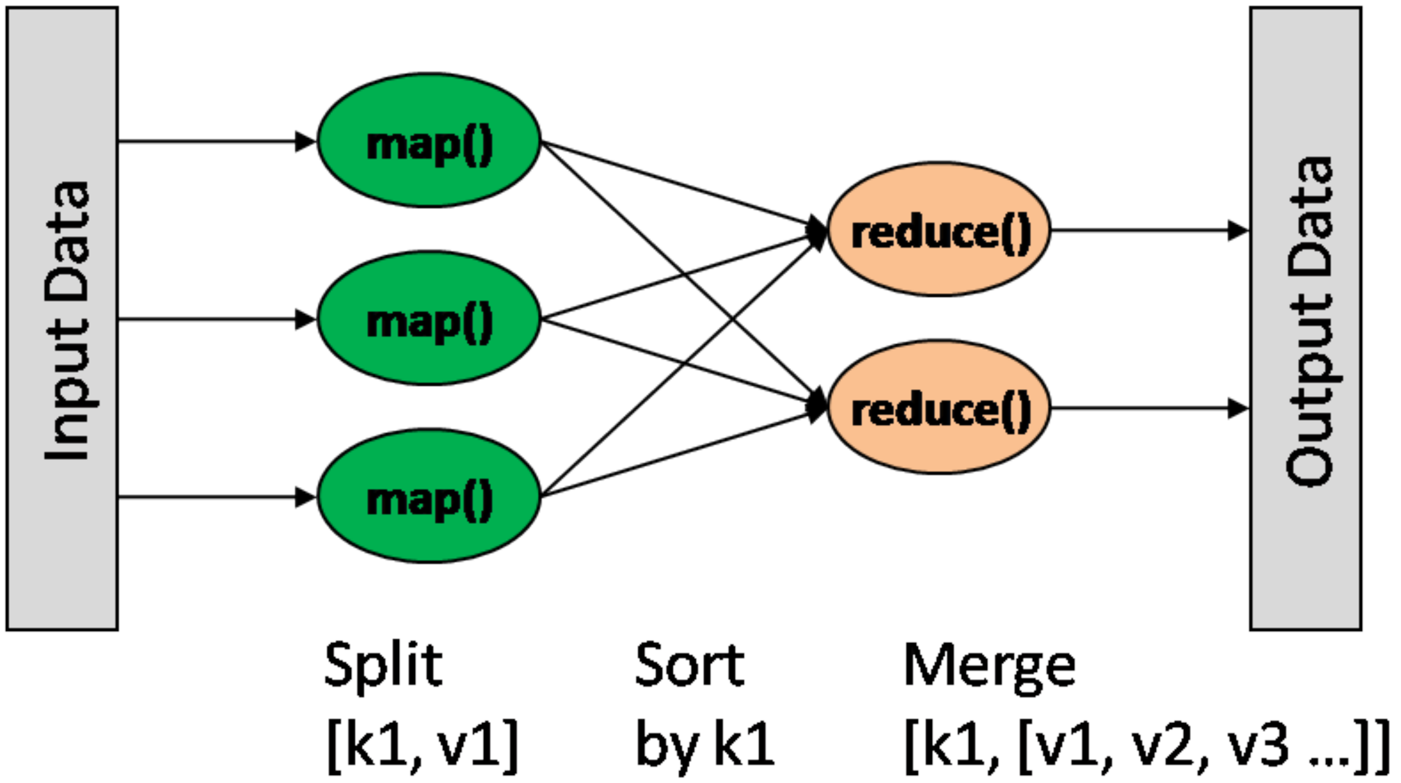
Condor HDFS: II

File transfer support

transfer_input_files = hfds://...

Spool in hdfs

Map Reduce


Input Data
map()
map()
map()
reduce()
reduce()
Output Data
Split
[k1, v1]
Sort
by k1
Merge
[k1, [v1, v2, v3 ...]]

Shell hackers map reduce

- grep tag input | sort | uniq -c | grep

MapReduce lingo for the native Condor speaker

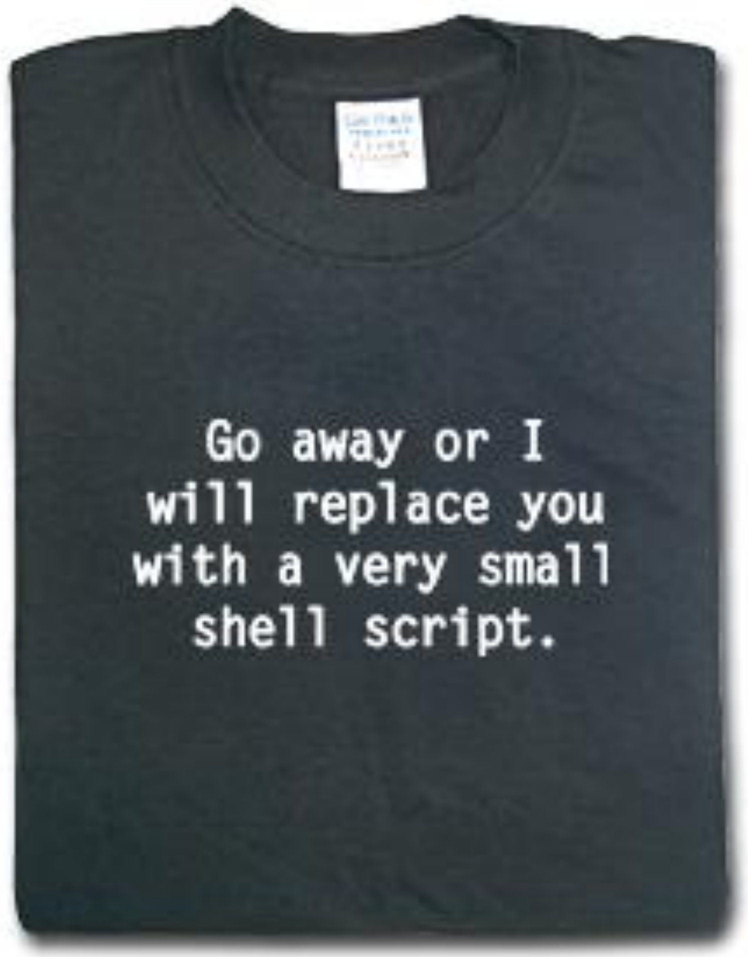
- Task tracker → startd/starter
- Job tracker → condor_schedd

Map Reduce under Condor

- Zeroth law of software engineering

- Job tracker/task tracker must be managed!
  - Otherwise very bad things happen

Hadoop on Demand w/Condor



Map Reduce as overlay

- Parallel Universe job
- Starts job tracker on rank 0
- Task trackers everywhere else
- Open Question:
  - Run more small jobs, or fewer bigger
- **One job tracker per user (i.e. per job)**

On to real science…

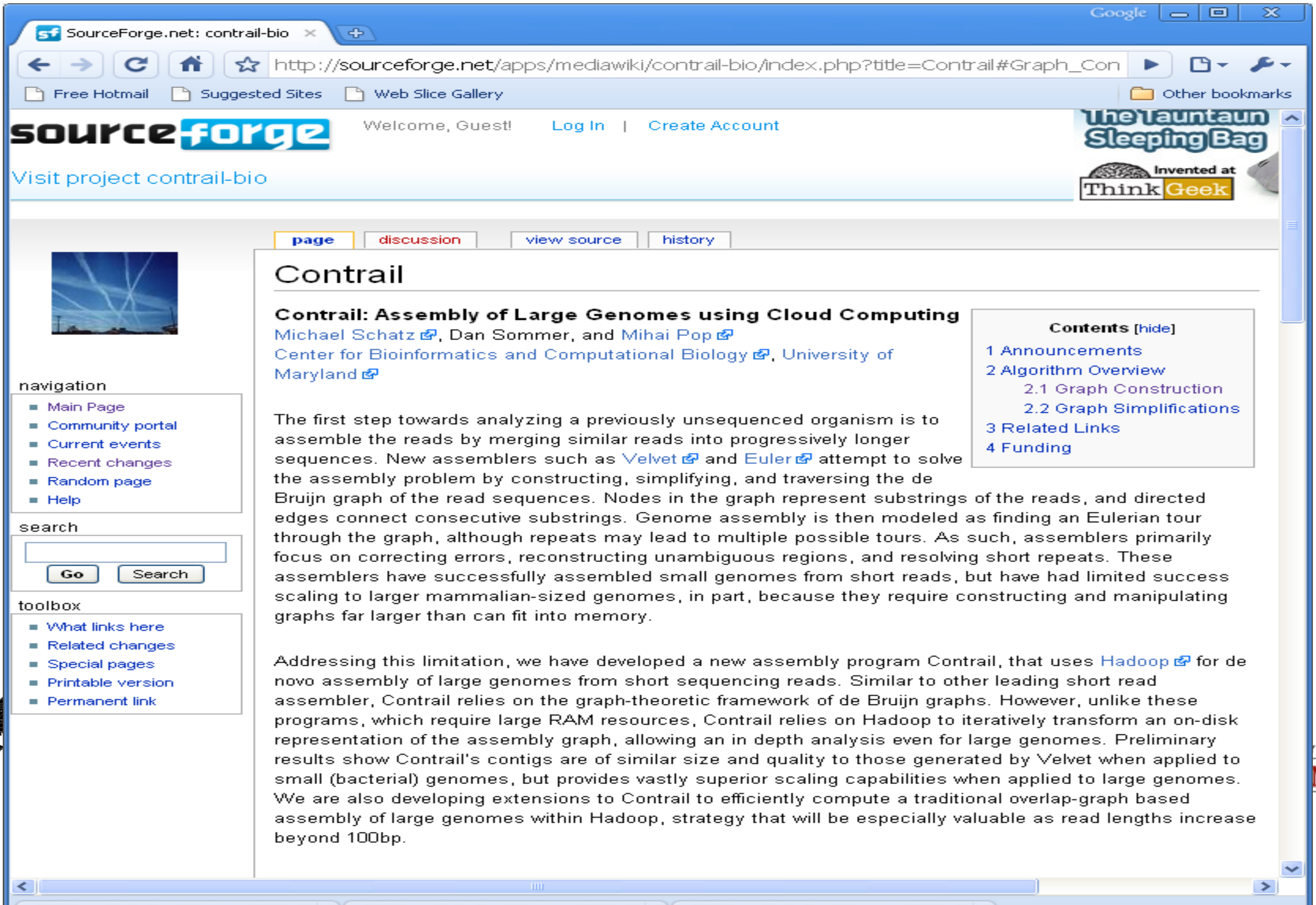
- David Schwartz, matchmaker

Mihai Pop

Contrail – MR genome assembly

http://sourceforge.net/apps/mediawiki/contrail-bio/index.php

Genome assembly



Contrail

Contrail: Assembly of Large Genomes using Cloud Computing

Michael Schatz, Dan Sommer, and Mihai Pop
Center for Bioinformatics and Computational Biology, University of Maryland

Contents [hide]

The first step towards analyzing a previously unsequenced organism is to assemble the reads by merging similar reads into progressively longer sequences. New assemblers such as Velvet and Euler attempt to solve the assembly problem by constructing, simplifying, and traversing the de Bruijn graph of the read sequences. Nodes in the graph represent substrings of the reads, and directed edges connect consecutive substrings. Genome assembly is then modeled as finding an Eulerian tour through the graph, although repeats may lead to multiple possible tours. As such, assemblers primarily focus on correcting errors, reconstructing unambiguous regions, and resolving short repeats. These assemblers have successfully assembled small genomes from short reads, but have had limited success scaling to larger mammalian-sized genomes, in part, because they require constructing and manipulating graphs far larger than can fit into memory.

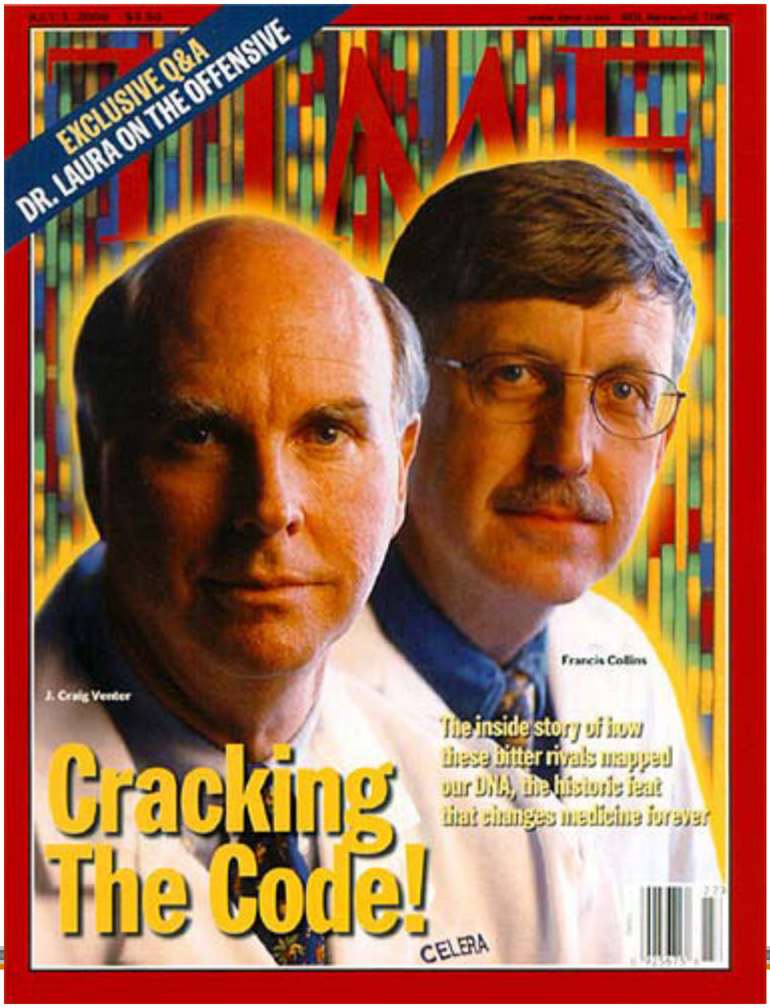
Addressing this limitation, we have developed a new assembly program Contrail, that uses Hadoop for de novo assembly of large genomes from short sequencing reads. Similar to other leading short read assembler, Contrail relies on the graph-theoretic framework of de Bruijn graphs. However, unlike these programs, which require large RAM resources, Contrail relies on Hadoop to iteratively transform an on-disk representation of the assembly graph, allowing an in depth analysis even for large genomes. Preliminary results show Contrail's contigs are of similar size and quality to those generated by Velvet when applied to small (bacterial) genomes, but provides vastly superior scaling capabilities when applied to large genomes. We are also developing extensions to Contrail to efficiently compute a traditional overlap-graph based assembly of large genomes within Hadoop, strategy that will be especially valuable as read lengths increase beyond 100bp.

DNA

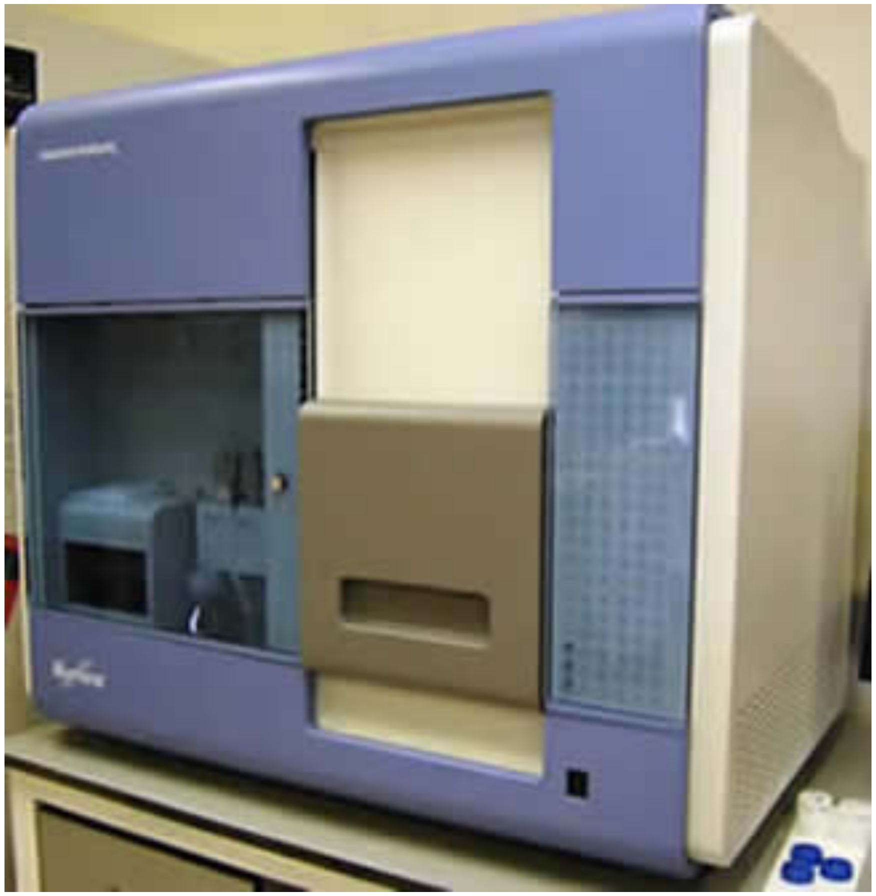
3 Billion base pairs

Sequencing machines only read small reads at a time

Already done this?


TIME
EXCLUSIVE Q&A
DR. LAURA ON THE OFFENSIVE
Francis Collins
J. Craig Venter
The inside story of how these bitter rivals mapped our DNA, the historic feat that changes medicine forever
Cracking The Code!
CELERA

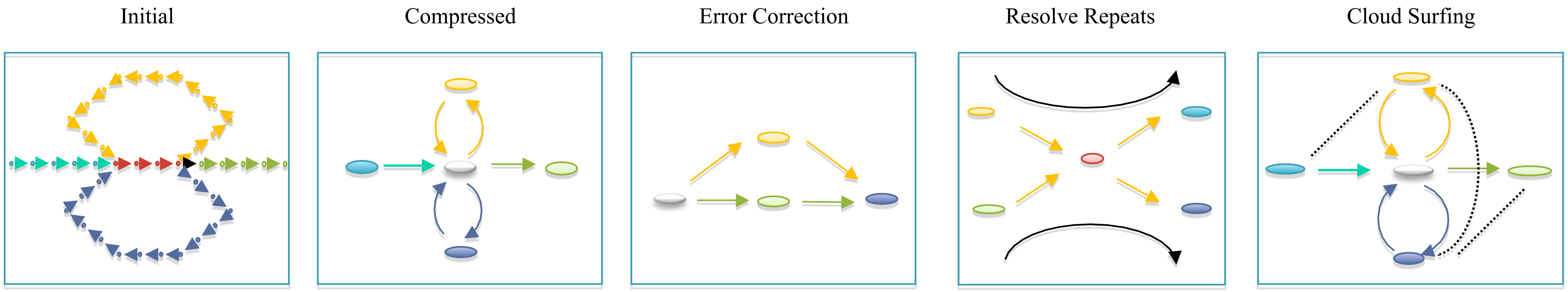
High throughput sequencers



Contrail

Scalable Genome Assembly with MapReduce

- *Genome:* African male NA18507 (Bentley *et al.*, 2008)
- *Input: 3.5*B 36bp reads, 210bp insert (SRA000271)
- *Preprocessor*: Quality-Aware Error Correction



	Initial	Compressed	Error Correction	Resolve Repeats	Cloud Surfing
N	>10B	>1 B	5.0 M	4.2 M	In Progress
Max	27	303 bp	14,007	20,594	
N50	27	< 100 bp	650 bp	923 bp	

Running it under Condor

- Used CHTC B-240 cluster
- ~100 machines
  - 8 way nehalem cpu
  - 12 Gb total
  - 1 disk partition dedicated to HDFS
  - HDFS running under condor master

Running it on Condor

- Used the MapReduce PU overlay
- Started with Fruit Flies
- …
- And it crashed
- Zeroth law of software engineering
    - Version mismatch
- Debugging…

Debugging

- After a couple of debugging rounds
- Fruit Fly sequenced!!
  - On to humans!

Cardinality

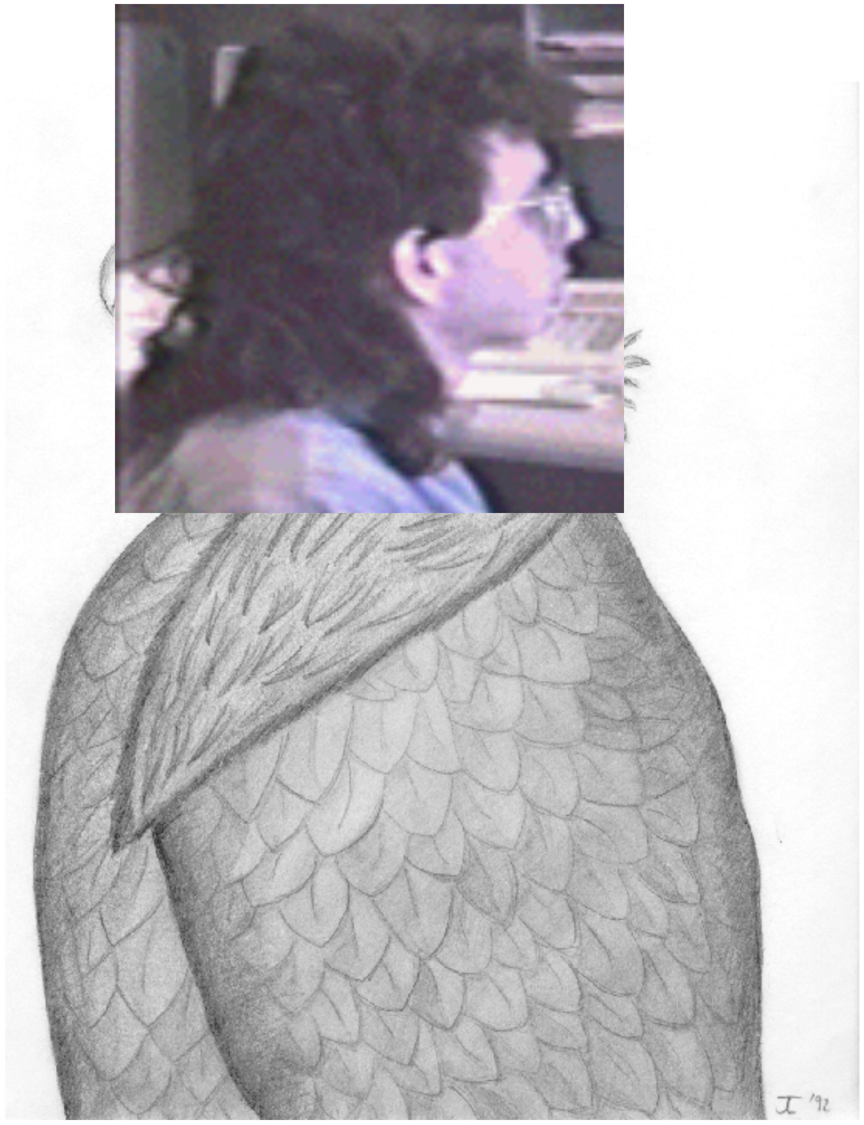
- How many slots per task tracker?
  - Task tracker, like schedd multi-slots
- One machine
  - 8 cores
  - 1 disk
  - 1 memory system
- How many mappers per slot

More MR under Condor

- More debugging, NPEs
- Updated MR again
- Some performance regressions
- One power outage

- 12 weeks later…

Success!


I '92

Conclusions

- Job trackers must be managed!
  - Glide-in is more than Condor on batch
- Hadoop – more than just MapReduce
- HDFS – good partner for Condor
- All this stuff is moving fast