Apache Hadoop framework
and Bioinformatics

Málaga, Julio 2011
Nacho Medina
imedina@cipf.es
http://bioinfo.cipf.es/imedina

Bioinformatics and Genomics Department
Centro de Investigación Príncipe Felipe (CIPF)
(Valencia, Spain)
Index

- Introduction
- Hadoop framework
  - HDFS
  - MapReduce
  - HBase
  - Others: Zookeeper, Avro, Pig, Mahout, …
- Bioinformatic applications
- Who is using it?
- Conclusions
Introduction
What is it Hadoop?

- It's a Java framework library that allows for the distributed processing of *large data sets* across clusters of computers using a simple programming model.

- Develops open-source software for reliable, scalable, distributed computing.

- Hadoop framework overview:

The project includes these subprojects:

- **Hadoop Common**: The common utilities that support the other Hadoop subprojects.
- **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
- **Hadoop MapReduce**: A software framework for distributed processing of large data sets on compute clusters.

Other Hadoop-related projects at Apache include:

- **Avro™**: A data serialization system.
- **Cassandra™**: A scalable multi-master database with no single points of failure.
- **Chukwa™**: A data collection system for managing large distributed systems.
- **HBase™**: A scalable, distributed database that supports structured data storage for large tables.
- **Hive™**: A data warehouse infrastructure that provides data summarization and ad hoc querying.
- **Mahout™**: A Scalable machine learning and data mining library.
- **Pig™**: A high-level data-flow language and execution framework for parallel computation.
- **ZooKeeper™**: A high-performance coordination service for distributed applications.
Introduction
Who is developing hadoop?

Apache Hadoop Project Members

Please see individual sub-projects for lists of their committers. We only here list the members of the top-level Hadoop project.

We ask that you please do not send us emails privately asking for support. We are non-paid volunteers who help out with the project and we do not necessarily have the time or energy to help people on an individual basis. Instead, we have setup mailing lists for each sub-project which often contain hundreds of individuals who will help answer detailed requests for help. The benefit of using mailing lists over private communication is that it is a shared resource where others can also learn from common mistakes and as a community we all grow together.

Hadoop PMC

The Hadoop Project Management Committee contains (in alphabetical order):

<table>
<thead>
<tr>
<th>username</th>
<th>name</th>
<th>organization</th>
<th>roles</th>
<th>timezone</th>
</tr>
</thead>
<tbody>
<tr>
<td>acmurthy</td>
<td>Arun C Murthy</td>
<td>Yahoo</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>cdouglas</td>
<td>Chris Douglas</td>
<td>Yahoo</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>cutting</td>
<td>Doug Cutting</td>
<td>Cloudera</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>ddas</td>
<td>Devaraj Das</td>
<td>Yahoo</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>drubha</td>
<td>Druba Sorthaiur</td>
<td>Facebook</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>eili</td>
<td>Elz Collins</td>
<td>Cloudera</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>enis</td>
<td>Enis Soztutar</td>
<td>Agrilab</td>
<td>+2</td>
<td></td>
</tr>
<tr>
<td>haihong</td>
<td>Haihong Kuang</td>
<td>Facebook</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>ianh</td>
<td>Ian Holman</td>
<td>AOL</td>
<td>+10</td>
<td></td>
</tr>
<tr>
<td>jghoman</td>
<td>Jakob Homan</td>
<td>LinkedIn</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>mahadev</td>
<td>Mahadev Konar</td>
<td>Yahoo</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>nigel</td>
<td>Nigel Dalay</td>
<td>Proximal Labs</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>o'malley</td>
<td>Owen O'Malley</td>
<td>Yahoo</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>phunt</td>
<td>Patrick Hunt</td>
<td>Cloudera</td>
<td>ZooKeeper</td>
<td>-8</td>
</tr>
<tr>
<td>rangadi</td>
<td>Raghu Angadi</td>
<td>Apple</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>shv</td>
<td>Konstantin Shvachko</td>
<td>eBay</td>
<td>HDFS</td>
<td>-8</td>
</tr>
<tr>
<td>sradia</td>
<td>Sanjay Rada</td>
<td>Yahoo</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>stack</td>
<td>Michael Stack</td>
<td>StumbleUpon</td>
<td>HBase</td>
<td>-8</td>
</tr>
<tr>
<td>suresh</td>
<td>Suresh Srinivas</td>
<td>Yahoo</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>sztszwo</td>
<td>Tsz Wo (Nicholas) Sze</td>
<td>Yahoo</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>todd</td>
<td>Todd Lipcon</td>
<td>Cloudera</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>tomwhite</td>
<td>Tom White</td>
<td>Cloudera</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>zshao</td>
<td>Zhang Shao</td>
<td>Facebook</td>
<td>-8</td>
<td></td>
</tr>
</tbody>
</table>

Emeritus Hadoop PMC Members

- Andrzej Bialecki
- Christophe Taton
- Jim Kellemann
Introduction
Why Hadoop? Data in bioinformatics

- From microarrays to NGS, a high-throughput paradigm change:
  - 1 microarray genotype experiment ~50MB => 1 NGS exome resequencing experiment ~50GB: **1000x** in data production!!
  - New experiments available: whole genome sequencing, about 1TB of data (~50TB of raw data) per experiment

- NGS analysis:
  - QC and preprocessing of few Terabytes of data (fastq format)
  - Reads Mapping
  - DNA Assembly
  - RNA-seq, ChIP-seq, Meth-seq, ...
Introduction
Why Hadoop? Other HPC solutions?

- Many different *HPC* solutions
  - MPI
  - Hadoop
  - GPU computing
  - ...

- No one is the “*best*” solution: *Analise your problem and choose the best solution for your specific problem, resources, mid-term goals, ...*

- Hadoop is aimed to process huge volumes of data of Tera or PetaBytes, what fits perfectly in many bioinformatics scenarios
Hadoop framework
HDFS, Hadoop Distributed Filesystem

- Provides high throughput access to application data and is suitable for applications that have large data sets
- Relaxes a few POSIX requirements to enable streaming access to file system data
- It's highly fault-tolerant and is designed to be deployed on low-cost hardware

Some Assumptions and goals:
- **Hardware failure**
- **Streaming data access**
  - the emphasis is on high throughput of data access rather than low latency of data access
- **Large data sets**
  - Terabytes of data. Tuned to support large files
- **Simple coherence model:**
  - A file once created, written, and closed need not be changed. A MapReduce application or a web crawler application fits perfectly with this model
- **Portability across heterogeneous hardware and software platform**

HDFS Architecture

- **Client**
  - Reads from **Datanodes**
  - Writes to **Datanodes**
- **Namenode**
  - Manages metadata
  - /home/foo/data, 3, ...
- **Replication**
  - Block replication across **Rack 1** and **Rack 2**
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 datasets. 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.

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

2.1 Example

Consider the problem of counting the number of occurrences of each word in a large collection of documents. The user would write code similar to the following pseudo-code:

```java
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

The `map` function emits each word plus an associated count of occurrences (just ‘1’ in this simple example). The `reduce` function sums together all counts emitted for a particular word.
Hadoop framework
Hadoop MapReduce implementation

- Hadoop MapReduce implementation is a software framework for easily writing applications which process vast amounts of data (*multi-terabyte data-sets*) in-parallel on large clusters (*thousands of nodes*) of commodity hardware in a reliable, fault-tolerant manner.

- A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework *sorts the outputs* of the *maps*, which are then input to the *reduce* tasks. The framework takes care of *scheduling tasks*, *monitoring them and re-executes* the failed tasks.

- *Typically the compute nodes and the storage nodes are the same*, that is, the MapReduce framework and the HDFS are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

- The MapReduce framework consists of a single master *JobTracker* and one slave *TaskTracker* per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.
JobTracker, to which client applications submit MapReduce jobs. The JobTracker pushes work out to available TaskTracker nodes in the cluster, striving to keep the work as close to the data as possible. With a rack-aware filesystem, the JobTracker knows which node contains the data, and which other machines are nearby.
**Hadoop framework**

**Hadoop MapReduce simple example**

- *WC*: word count example

- Only need to implement to methods:
  - `map()`
  - `reduce()`
Hadoop framework
Bigtable from Google 2006

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,wilsonh,kerr,m3b,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 high availability. Bigtable is used by more than sixty Google products and projects, including Google Analytics, Google Finance, Orkut, Personalized Search, WriteIt, and Google Earth. These prod-

• 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
• No SQL schema, different number of columns is normal, very flexible
Hadoop framework
HBase, a Bigtable implementation

Users store data rows in labelled tables. A data row has a **sortable** key and an **arbitrary** number of columns. The table is stored sparsely, so that rows in the same table can have crazily-varying columns, if the user likes.

**Example:**

- Querying for the row/column of "aaaaa"/"A:foo" will return "y"
- while querying for the row/column/timestamp of "aaaaa"/"A:foo"/10 will return "m"
- Querying for a row/column/timestamp of "aaaaa"/"A:foo"/2 will return a null result
Hadoop framework
HBase, some tutorials

- Good tutorial to understand HBase/BigTable

- Performance options
Avro: a compact, fast and binary data serialization system

Avro specifies two serialization encodings: **binary** and **JSON**.

Most applications will use the binary encoding, as it is smaller and faster.

But, for debugging and web-based applications, the JSON encoding may sometimes be appropriate.
Hadoop framework

Other related projects

- **Hive**: facilitates easy data summarization, ad-hoc queries, and the analysis of large datasets stored in Hadoop compatible file systems.

Example:

```sql
>>CREATE TABLE u_data (
  userid INT,
  movieid INT,
  rating INT,
  unixtime STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '	'
STORED AS TEXTFILE;
```

```sql
>>LOAD DATA LOCAL INPATH 'ml-data/u.data'
OVERWRITE INTO TABLE u_data;
```

```sql
>>SELECT COUNT(*) FROM u_data;
```
Hadoop framework
Other related projects

- **Mahout**: Scalable machine learning and data mining

**What is Apache Mahout?**

The Apache Mahout™ machine learning library's goal is to build scalable machine learning libraries.

With scalable we mean:

Scalable to reasonably large data sets. Our core algorithms for clustering, classification and batch based collaborative filtering are implemented on top of Apache Hadoop using the map/reduce paradigm. However we do not restrict contributions to Hadoop based implementations: Contributions that run on a single node or on a non-Hadoop cluster are welcome as well. The core libraries are highly optimized to allow for good performance also for non-distributed algorithms.

Scalable to support your business case. Mahout is distributed under a commercially friendly Apache Software license.

Scalable community. The goal of Mahout is to build a vibrant, responsive, diverse community to facilitate discussions not only on the project itself but also on potential use cases. Come to the mailing lists to find out more.

**Mahout currently has**
- Collaborative Filtering
- User and Item based recommenders
- K-Means, Fuzzy K-Means clustering
- Mean Shift clustering
- Dirichlet process clustering
- Latent Dirichlet Allocation
- Singular value decomposition
- Parallel frequent pattern mining
- Complementary Naive Bayes classifier
- Random forest decision tree based classifier
- High performance java collections (previously col collections)
- A vibrant community
- and many more cool stuff to come by this summer thanks to Google summer of code

**Classification**
A general introduction to the most common text classification algorithms can be found at [Google Answers](http://answers.google.com/answers/main?omHt=hsa&ai=1&rs=15&ht=25&st=2006). For information on the algorithms implemented in Mahout (or scheduled for implementation) please visit the following papers:

- Logistic Regression (LDA)
- Bayesian
- Support Vector Machines (SVM) (open: MAHOUT-31, MAHOUT-34 and MAHOUT-35)
- Perceptron and Vowpal Wabbit (open: MAHOUT-25)
- Neural Network (open, but MAHOUT-26 might help)
- Random Forests (integrated: MAHOUT-122, MAHOUT-140, MAHOUT-145)
- Restricted Boltzmann Machines (open: MAHOUT-375, GLOM(2010))
- Online Passive Aggressive (learning patch context): MAHOUT-702

**Clustering**

- Reference Reading
  - Copacabana Clustering (MAHOUT-3 - integrated)
  - k-Means Clustering (MAHOUT-5 - integrated)
  - Fuzzy k-Means (MAHOUT-4 - integrated)
  - Expectation Maximization (EM) (MAHOUT-20)
  - Mean Shift Clustering (MAHOUT-5 - integrated)
  - Hierarchical Clustering (MAHOUT-13)
  - Dirichlet Process Clustering (MAHOUT-30 - integrated)
  - Latent Dirichlet Allocation (MAHOUT-123 - integrated)
  - Spectral Clustering (MAHOUT-363 - integrated)

**Pattern Mining**

- Parallel FP Growth Algorithm (also known as frequent itemset mining)

**Regression**

- Local Weighted Linear Regression (open)

**Dimension reduction**

- Singular Value Decomposition and other Dimension Reduction Techniques (available since 0.3)
- Principal Component Analysis (PCA) (open)
- Independent Component Analysis (open)
- Gaussian Discriminant Analysis (GDA) (open)

**Evolutionary Algorithms**

- see also: MAHOUT-GA (integrated)

You will find here information, examples, use cases, etc. related to Evolutionary Algorithms.

**Introductions and Tutorials:**
- Evolutionary Algorithms Introduction
- How to distribute the fitness evaluation using Mahout/GA

**Examples:**
- Traveling Salesman
- Class Discovery
Hadoop framework

Other related projects

- **Pig**: A high-level data-flow language and execution framework for parallel computation

**Example**:

```java
>> A = LOAD 'student' USING PigStorage() AS (name:chararray, age:int, gpa:float);
>> B = FOREACH A GENERATE name;
>> DUMP B;
```
Hadoop framework
Other related projects

- **Hama**: a distributed computing framework based on *BSP* (*Bulk Synchronous Parallel*) computing techniques for massive scientific computations.
Hadoop framework
Other related projects

- **Zookeeper**: is a distributed, open-source coordination service for distributed applications

---

Welcome to Apache ZooKeeper™

Apache ZooKeeper is an effort to develop and maintain an open-source server which enables highly reliable distributed coordination.

**What is ZooKeeper?**

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

Learn more about ZooKeeper on the [ZooKeeper Wiki](https://zookeeper.apache.org/).
The Genome Analysis Toolkit (GATK) is a structured programming framework designed to ease the development of efficient and robust analysis tools for next-generation DNA sequencing data produced using the functional program philosophy of MapReduce. The GATK provides a small but rich set of data access patterns that encompass the majority of analysis tool needs. Separating specific analysis calculations.

http://www.broadinstitute.org/gsa/wiki/index.php/The_Genome_Analysis_Toolkit
Bioinformatic applications
Bowtie Crossbow, genome resequencing


Crossbow is a scalable software pipeline for whole genome resequencing analysis. It combines Bowtie, an ultrafast and memory efficient short read aligner, and SoapSNP, and an accurate genotyper. These tools are combined in an automatic, parallel pipeline that runs in the cloud (Elastic MapReduce in this case) on a local Hadoop cluster, or on a single computer, exploiting multiple computers and CPUs wherever possible. The pipeline can analyze over 35x coverage of a human genome in one day on a 10-node local cluster, or in 3 hours for about $85 using a 40-node, 320-core cluster rented from Amazon Web Services.

Version 1.1.2 - May 23, 2011
- Added --just-align and --resume-align options. --just-align causes Crossbow to put the results of the Alignment phase in the output directory and quit after the
Bioinformatic applications
CloudBurst, a NGS read mapping


CloudBurst: highly sensitive read mapping with MapReduce.

Schatz MC.
Center for Bioinformatics and Computational Biology, University of Maryland, College Park, MD 20742, USA. mschatz@umiacs.umd.edu

Abstract

MOTIVATION: Next-generation DNA sequencing machines are generating an enormous amount of sequence data, placing unprecedented demands on traditional single-processor read-mapping algorithms. CloudBurst is a new parallel read-mapping algorithm optimized for mapping next-generation sequence data to the human genome and other reference genomes, for use in a variety of biological analyses including SNP discovery, genotyping and personal genomics. It is modeled after the short-read-mapping program RMAT, and reports either all alignments or the unambiguous best alignment for each read with any number of mismatches or differences. This level of sensitivity could be prohibitively time consuming, but CloudBurst uses the open-source Hadoop implementation of MapReduce to parallelize execution using multiple compute nodes.

RESULTS: CloudBurst's running time scales linearly with the number of reads mapped, and with near linear speedup as the number of processors increases. In a 24-processor core configuration, CloudBurst is up to 30 times faster than RMAT executing on a single core, while computing an identical set of alignments. Using a larger remote compute cloud with 96 cores, CloudBurst improved performance by ~100-fold, reducing the running time from hours to mere minutes for typical jobs involving mapping of millions of short reads to the human genome.

AVAILABILITY: CloudBurst is available open-source as a model for parallelizing algorithms with MapReduce at (http://cloudburst-bio.sourceforge.net).
Bioinformatic applications
Contrail, a distributed DNA assembler


Contrail: Assembly of Large Genomes using Cloud Computing
Michael Schatz, Dan Sommer, David Kelley, and Mihai Pop
Center for Bioinformatics and Computational Biology, University of Maryland

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 assemblers, 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.

Contrail enables de novo assembly of large genomes from short reads by bridging research in computation biology with research in high performance computation. This combination is essential in light of the large data sets involved, and has the potential to unlock discoveries of critical magnitude. Whereas the published analysis of the African and Asian human individuals used reads mapping to discover conserved regions and regions with small polymorphisms, de novo assembly has the unique potential to also discover large scale polymorphisms between these individuals and the reference human genome. Mapping the large-scale differences is an important step towards better understanding of our own biology, and may reveal previously unknown characteristics of the human genome related to health or disease. Furthermore, a short read assembler for large genomes is also essential for sequencing the vast numbers of complex organisms that have never been sequenced before, and will directly contribute to new biological knowledge.
Who is using it?
Amazon Elastic MapReduce

http://aws.amazon.com/elasticmapreduce/

Amazon Elastic MapReduce is a web service that enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data. It utilizes a hosted Hadoop framework running on the web-scale infrastructure of Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3).

Using Amazon Elastic MapReduce, you can instantly provision as much or as little capacity as you like to perform data-intensive tasks for applications such as web indexing, data mining, log file analysis, data warehousing, machine learning, financial analysis, scientific simulation, and bioinformatics research. Amazon Elastic MapReduce lets you focus on crunching or analyzing your data without having to worry about time-consuming set-up, management or tuning of Hadoop clusters or the compute capacity upon which they sit.
Who is using it?
CouchDB, a document-oriented database

Apache CouchDB is a document-oriented database:  http://couchdb.apache.org/

The Apache CouchDB Project

Apache CouchDB is a document-oriented database that can be queried and indexed in a MapReduce fashion using JavaScript. CouchDB also offers incremental replication with bi-directional conflict detection and resolution.

CouchDB provides a RESTful JSON API that can be accessed from any environment that allows HTTP requests. There are myriad third-party client libraries that make this even easier from your programming language of choice. CouchDB's built-in Web administration console speaks directly to the database using HTTP requests issued from your browser.

CouchDB is written in Erlang, a robust functional programming language ideal for building concurrent distributed systems. Erlang allows for a flexible design that is easily scalable and readily extensible.

See the introduction and the technical overview for more information.

How to Get Started

Read the introduction and the technical overview to learn more about the design and basic concepts of CouchDB. To give it a try, download the CouchDB code from the Subversion repository and follow the included build instructions.
Who is using it
Yahoo!, Facebook, ...

- **Yahoo!** has more than 100,000 CPUs in >40,000 computers running Hadoop
  - Our biggest cluster: *4500 nodes* (2*4cpu boxes w 4*1TB disk & 16GB RAM)
  - Used to support research for Ad Systems and Web Search
  - Also used to do scaling tests to support development of Hadoop on larger clusters
  - >60% of Hadoop Jobs within Yahoo are Pig jobs.

- **Facebook** use Hadoop to store copies of internal log and dimension data sources and use it as a source for reporting/analytics and machine learning. Currently we have 2 major clusters:
  - A *1100-machine cluster* with 8800 cores and about 12 PB raw storage.
  - A *300-machine cluster* with 2400 cores and about 3 PB raw storage.
  - Each node has 8 cores and 12 TB of storage.
Conclusions

- A complete framework for analysis and processing of large datasets
- Fits perfectly in many bioinformatics scenarios
- Many related projects available, complete and integrated solution