Function:Troilkatt/Design Document

From FunctionWiki

This document is used for the design description. See also the progress list, the design document for private data search, and meeting notes.

To go up.

NOTE! Document is under reorganization

Contents

- 1 Notes
- 2 Version History
- 3 Terminology
- 4 Overview
- 5 Requirements and Non-Requirements
- 6 Goals
- 7 Dependencies
- 8 Deliverables
- 9 Open Issues
- 10 Detailed Description
  - 10.1 Operational Scenario
  - 10.2 User Interface
  - 10.3 Architecture
  - 10.4 Performance Considerations
  - 10.5 Data Sources and Structure
  - 10.6 Processing Pipeline
  - 10.7 Data Storage
  - 10.8 Parallel Spell
  - 10.9 Possible Optimizations
  - 10.10 Private Data
  - 10.11 Failures
- 11 Programming Interface
  - 11.1 Processing Pipeline
  - 11.2 Parallel Spell
- 12 Evaluation
- 13 Issues to check and verify
- 14 Previous Work
- 15 Future Directions
- 16 Templates

Notes

Version History

- Version 0.1: [1] (http://incendio.princeton.edu/functionwiki/index.php?title=Function:Troilkatt

/Design_Document&oldid=2360)
- Version 0.2: [2] (http://incendio.princeton.edu/functionwiki/index.php?title=Function:Troilkatt /Design_Document&oldid=2604)
- Version 0.3: [3] (http://incendio.princeton.edu/functionwiki/index.php?title=Function:Troilkatt /Design_Document&oldid=4122)

Terminology

- Dataset: data from one experiment that can have many samples (.SOFT files from GEO, or MAGEML files from ArrayExpress)
- Compendium: integrated datasets.
- Processing or pre-processing: steps done to convert downloaded data to a format that can be integrated into a compendium.
- Gene query: Given a small set of query genes, SPELL (http://mmxserver.cs.princeton.edu:3000/search) identifies which datasets are most informative for these genes, then within those datasets additional genes are identified with expression profiles most similar to the query set.

Overview

Technology advances in instruments for genomics data have increased the number of and size of available data to such a scale that these can no longer be integrated and interactively searched and explored on a single system. The goal of this project is twofold. First to develop a system that supports automatic data retrieval and integration of most publicly available genomics datasets, and support interactive search in genomic data of millions of samples. Second, to deploy the system on a 66-node cluster hosted at Princeton University, and use it to provide interactive exploration of all published microarray datasets using the Spell approach.

Requirements and Non-Requirements

See also: progress list.

The requirements are split into three lists: must-have requirements, nice-to-have requirements, and non-requirements. The system design will take into account both must-, and nice-to-have requirements, but we will first implement it to satisfy the must-have requirements.

Must-have requirements:

1. Automatic retrieval of all microarray datasets from GEO and ArrayExpress.
2. Automatic processing and conversion of downloaded data to the PCL format for gene correlation calculation, and integration to a compendium (including Spell)
3. Manual addition, deletion, and updating of a dataset in a compendium.
4. Interactive Spell search for Human dataset with latency less than 5 seconds.
5. Fault tolerance such that the correct [or a partial?] result is returned in 5 seconds even if a node crashes during the query execution.
6. Support for users to integrate private data with existing compendium while maintaining strict privacy restrictions (e.g. for human patient data).
7. Spell search combining user submitted data and the public data already in the compendium.
8. Map-reduce approach used for implementation.

Nice-to-have:

1. Scalability to millions of datasets.
2. Automated logging and error reporting for download, conversion, and integration of datasets.
3. Reliable storage of meta-data and data to a allow data compendium be rebuilt by re-downloading, re-computing, and re-integrating datasets.

4. Save all previous versions of a compendium and allow users to access an old version of a compendium.
5. More than 3x compression ratio for all data, version based compression for compendium, duplicate elimination, and de-compression throughput > 10 Mbps/s.
6. Allow multiple users to search simultaneously.
7. Log query information for the later optimization.

Non-requirments:

1. Automatic download of meta-data such as list of genes in an organism and gold standards. There are no good sources for such lists, and in practice a human expert needs to modify these to achieve the best search results. Instead such lists must be manually maintained.
2. Automatic retrieval, management, and integration of aligned sequence datasets from Trace Archive. We do not have the processing pipeline necessary to convert these to PCL files <at least to my knowledge>?.
3. Support multi-organism compendium. This will significantly increase the number of genes, but requires more thinking of how to model this.

Goals

- Provide largest up-to-date collection of integrated microarray data.
- Scalability to handle storage and compute requirements of next generation integration algorithms.
- Interactive gene query that scales to tens-of-thousands of datasets and genes.
- Deployment of system.

Dependencies

We will build the system using:

- The hadoop file system, to leverage distributed storage.
- Hadoop MapReduce for fault tolerant parallel processing.
- Hbase for random access and data organization for large tables.
- GWT for implementing broweser user interface.
- Java Webstart for one-click start of private search program.
- Jetty for running private search program.

Deliverables

- Deployed automatic download system.
- Deployed parallel Spell.
- Collection downloaded, processed, and integrated microarray data.
- MySpell application/ plugin that allows for Spell search using private data.

Open Issues

Additional open issues are also discussed in the Detailed Description section. But these will be resolved once the design/implementation is in progress.

Detailed Description

Operational Scenario

The system is to be deployed on a 66-node cluster. It will weekly download new data from GEO and ArrayExpress, convert the downloaded files to a common format, calculate gene correlations, and integrate them with previously

downloaded data. Model developers can use the data stored on the cluster and the services provided by us to develop new models. End-users can search and explore the integrated datasets using Web based interfaces and the private data search program.

Initially we target a Human compendium composed of 724 datasets, 15.858 samples, 24.410 genes in the Human genome. However the system is designed to scale for much larger larger compendium.

User Interface

There are three types of users:

End-users are biologists using a modified Spell web interface, that provides:

1. The existing web-interface (input box for genes, and visual presentation of the results).
2. Existing interface extended with support for selecting a version of the compendium to search.
3. New interface for private data integration with out public data.

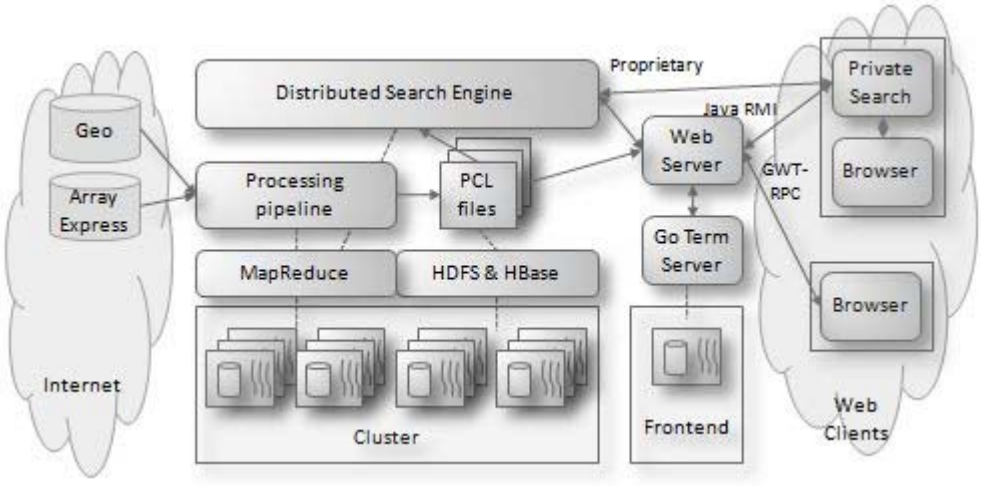
Maintainer is (are) persons responsible for maintaining the system, by using tools that provide:

1. Error logs for downloaded datasets that could not be converted, processed, or integrated.
2. Script based tools for viewing detailed logs for the processing of each dataset.
3. Scripts for adding, deleting, or updating a dataset in a compendium.
4. Scripts for manually adding a dataset to be processed and integrated.

Model developers are persons using the datasets and integrated data to develop new integration models. These use:

1. The programming API described below.
2. Scripts for downloading files maintained by our system.

Architecture



Troilkatt consists of six main parts:

- An automated *processing pipeline* downloads datasets from public repositories such as GEO and ArrayExpress, runs a set of processing steps on the datasets, and saves the converted datasets as PCL files. The pipeline saves uses Hadoop File Systems (HDFS) and Hbase for data storage, and Hadoop MapReduce for parallel data processing.
- A *distributed search engine* runs Spell queries in parallel on a cluster.
- A *Web server* exports a Google Web Toolkit (GWT) interface to browser clients and a Java RMI interface to private search clients.
- A *Go Term server* does meta-data search on Go Terms.
- *Browser GUI* implemented using GWT.

- A *private search program* that allows integrating private datasets with our public datasets.

Performance Considerations

The performance critical part of the system is Spell search component for user queries. File download, processing, and conversion are not performance critical. Instead the goal is to have reasonable resource usage. Especially important is to avoid significant perturbation of the search latency.

Data pre-processing is embarrassingly parallel and should not have any major scalability issues. Even the compute and data intensive processing to create the lookup tables can take advantage of the proven scalability of the MapReduce approach. Similarly, the data storage for the compendium depends on the scalability of HDFS and HBase. One major storage issue is reliable backup of the compendium. For these very efficient version based compression is needed.

Scalability of gene query part is maximum to dataset number since we take dataset as processing unit. According to this design decision, we only get partial gene score corresponding to one dataset instead of final gene scores. For human genome, there 1500 messages in 100KB size. Note if there are lots of zero weights, we can expect to reduce communication here. Message collecting for final scores is incline to be bottleneck.

Parallel search scalability is already discussed in Parallel Search

Notes:

- TODO: Amount of storage we can use. <not a requirement, but discussion>
- TODO: Amount of computation power we can use. <not a requirement, but...>

Data Sources and Structure

For more details, see Function:Troilkatt/Data_Sources See also not implemented data structures

Public repositories with genomic data.

Repository	GEO:Series	GEO:Datasets	ArrayExpress	Trace Assembly	Trace Archive	Short Read Archive
Compressed size	0.3TB	a few GB's	4.5TB?	0.1TB	2.5TB	12TB
Datasets	14.764	2089	9934	4154	5957	8406
Samples	379.898	?	?	?	?	

The above table lists the size, number of datasets, and total number of samples in all datasets. Note that for integration a distance matrix is usually calculated for all samples in a dataset.

Currently only the microarray repositories (GEO and ArrayExpress) are downloaded since we do not yet have a processing pipeline for the sequence data. But only data from GEO are integrated since we do not have an ArrayExpress pre-processing pipeline. All downloaded data is saved in HDFS as described in section Data Storage.

The format of the downloaded files is determined by the remote repository (more details Function:Troilkatt/Data Sources). These formats may be hard to work with so they are converted to the simple **PCL** tabular text file format, that contains a table with one row per gene, and one column per experiment.

PCL File Content.

ORF	NAME	GWEIGHT	Description 1	Description 2	...	Description E
EWEIGHT			**1**	**1**	**...**	**1**
Gene1	**Gene1**	**1**	**expression_1_1**	**expression_1_2**	**...**	**expression_1_E**

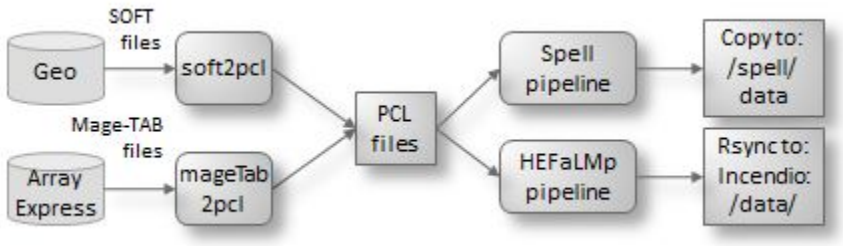
Gene1	Gene1	1	expression_2_1	expression_2_2	...	expression_1_E
...					...	
GeneN	GeneN	1	expression_N_1	expression_N_2	...	expression_N_E

For Human there are currently 4995 series files in GEO, and 724 dataset files. We have not counted the number of columns in each file, but if we take the GEO average of 25.7 columns per file we get that these have respectively 128372 and 18068 columns. **Note!** that these have not yet been run through the pre-processing pipeline so there may be datasets that cannot be integrated. Most datasets are less than 2MB (see GEO for details).

It is possible to pre-calculate the gene-to-gene distance matrices. However, the distance matrices takes up to three order of magnitude more storage than the raw data, and the time to read a row from disk is about the same as the time to calculate the distance matrices. We therefore do the distance calculation at runtime.

Processing Pipeline

For more details, see Function:Troilkatt/Pipeline



A Troilkatt pipeline is used to do a series of processing steps on downloaded data. A pipeline has a source, a set of processing stages, and a sink. The source can be a public data repository as Geo or ArrayExpress, a sink from another pipeline, or a directory on the local filesystem. Each stage in the pipeline takes a set of files, does some processing on them, and outputs a set of files. The sink is used to save a list of files that are useful either as input to another pipeline, or as the final output of the conversion pipeline.

The processing pipelines has been designed to be backward compatible with previously written scripts and tools. It moves all the scripts, meta-data files, and knowledge about the processing order from individual developers personal computers to a central repository.

Before the downloaded files can be integrated to a compendium they are converted from the downloaded format into PCL format, and then the PCL files are sent through a pipeline with several computation stages each implementing some data cleanup algorithm. In the pipeline the first stage is always source and the last is a sink. Each stage takes lists of new, updated, deleted, does some calculation on the all files and produces similar lists of files as output. A stage can be executed in parallel, using MapReduce, by having each mapper doing calculation on one file.

An approach such as hadoop sequence cannot be used since the scripts/tools typically read in an entire table from a file before processing it. In addition many program arguments such as filenames and directories are replaced with symbols to allow the system to execute the scripts in parallel and to use library of executables and meta-data files.

Data Storage

Data is stored on the cluster node local file systems, NFS, HDFS, and Hbase. The local filesystem is used to temporarily save files downloaded from public repositories, and files being processed by external scripts and tools. This is necessary since these are written to access generic file system API that is currently not supported by HDFS. A common operation during the data processing is therefore to move files to and from HDFS. The operating system should optimize this, and the files are typically only a few to tens of megabyte in size.

NFS is used to store log-files, meta-data files, and executables. For all of these it having a single filesystem is practical

and neither has high performance or data size requirements.

HDFS is used to save all downloaded, intermediate, and processed files. We rely on the large capacity and parallel data processing for the data integration part. HBase is used for the large lookup tables, and the meta-data structures. For the lookup table the motivation is to take advantage of the random access properties and column based compression, while for the meta-data it is to take advantage of the timestamped data organization.

One of the requirements is to provide search repeatability for biology researcher, the system must guarantee that the same compendium is used for the search. HDFS files and HBase tables are stored on the cluster nodes with three replicas. We do not have the capacity to reliably back up everything. In case of a catastrophic failure it is possible to re-download files from the public repositories and re-compute the compendiums. However, it will be very hard to do the computation under the exactly same conditions since there may be changes to the scripts and tools, meta-data, and libraries. We therefore back up the final output of the compendium on reliable storage. The compendium can then be restored in case of catastrophic failure, ensuring that searches on the old versions return identical results. Neither the source or intermediate files are saved, so to update the compendium all source files must be re-downloaded and re-integrated. An exception are source files or intermediate files changed by the maintainer. These are also saved reliability and later added to the compendia.

New and updated files are downloaded from the public repositories once a week (with more frequent updates there will be more versions that may take slightly more storage space, but we do not believe computation time will be an issue). The new and updated files are sent through the processing pipelines as described above, and the compendiums are updated with these files as described above. Everything is timestamped. The timestamp is set when the main processing loop starts such that all files downloaded, all processing intermediate files, and all compendium has the same timestamp. A read can return either the latest version of the compendium, or a specific version.

All intermediate files created during data processing are saved in HDFS. These are useful for the system maintainer (or developer) to check in case a file could not be converted or integrated. However, these files can periodically be deleted for example after 2-3 months.

Parallel Spell

See also: modified Spell protocol

The overview of the Spell algorithm is as follows:

1. The user provides a query set of genes of interest. For example these can be genes known to be related to a certain disease.
2. The system reads in a set of files containing processed datasets.
3. Each dataset is assigned a relevance weight based on the gene-to-gene distance between the query genes in the dataset.
4. Given these weights for each dataset, a per-gene score is calculated as the weighted correlations to the query set for all genes across all datasets in the compendium.
5. The search returns the datasets most relevant to the (biological context defined by the) query genes and identifies additional genes related in these datasets.

The parallel implementation of Spell divides datasets among tasks. Each task reads all samples from it's dataset and correlates these to get query-gene to gene distances. Then each task calculates the dataset weight and partial gene scores for it's dataset. To get the score for a gene the partial score for the gene calculated by each tasks must be gathered and added together. The final scores are then sorted and returned to the user.

We also considered alternative design alternatives where the gene-to-gene distances are pre-calculated. These require two or three orders of magnitude more storage, and does not significantly improve performance since the time to read the pre-computed values from disk is about the same as computing these. Note that it is not possible to pre-compute the final gene scores since they depend on the dataset weights, which in turn depends on genes in the query. It is therefore only possible to pre-compute the gene-to-gene distances for each dataset.

The parallel search should be run within the hadoop framework, but not necessarily as a MapReduce job, to take advantage of processes being mapped to the nodes where the data is, fault tolerance, and counters etc.

Possible Optimizations

Users can *refine* a search by adding high scoring genes to the query set. This offers a possibility to cache previously read or calculated results to reduce search latency. Thus, it is only necessary to calculate the gene scores and dataset weights for the genes added or removed from the query.

If the search time latency requirement cannot be met, it is possible to split the result presentation into three parts:

1. Datasets ordered according to weight (these are the column headers).
2. Genes ordered based on the scores for the top N datasets.
3. Genes ordered based on the scores for all datasets.

The staging complicates the implementation (issues):

- All dataset weights should be calculated before any of the gene scores are calculated.
- The gene score calculation must be scheduled such that the top N datasets are calculated first.

Private Data

For more details see: Function:Troilkatt/Private Data

Genomic datasets that include Human data may be under very strict privacy and confidentially laws that prevents the data to be uploaded to a cloud for analysis. It is therefore required to provide:

1. Spell search where users can integrate private data with existing compendium while maintaining strict privacy restrictions (e.g. for human patient data). That is:
    - Keep all raw and processed data on the user's machine.
    - Do not send any information about the data, including summary of query results.
2. Provide all code to be executed on the user's machine as a single-click run plugin/application.

Failures

There are four types of failures that may occur:

1. The system crashes under data downloading, processing, or integration. This may be due to bugs in the Troilkatt code, one of the Hadoop systems crashing (that is Hbase), or the cluster being shut down or rebooted.
2. Hbase crashes and data is lost. This is possible since HDFS does not support append.
3. One of the cluster nodes crashing, including loosing one of the disks.
4. Catastrophic failure with many cluster nodes being destroyed, such that most of the data on the cluster is lost.

The first case is common, especially during the development and testing of the system. However, all processing is deterministic, so in case of a crash the processing can be re-run to produce the missing results. To detect such crashes two strategies are used. First, at the start of the main Troilkatt loop a special files is updated by writing status as *timestamp: updating*. This file is then saved to HDFS before starting any processing. Once all processing is done, the status is reset o *timestamp: done*. Crashes are detected by checking if the status is *updating* at the beginning of the loop. If so, the last update in the pipeline table is checked against the current timestamp. If it matches, the stage completed and does not need to be re-executed. This is possible since the last operation of each stage is to update the pipeline table.

The second case is similar to the first case unless data loss is out-of-order. That is if all meta-data from a given stage is loss then the stages will be re-executed, but if status update is completed then the system will not detect any failures. <One possible solution is to check if all stages completed in the previous execution>. TODO: understand exactly what may happen

The third case should for the data processing be handled by data replication. For search, a faulty node may significantly delay the job. In this case the query handled by the faulty/slow nodes is re-executed on other nodes. Since the query time depends on the number of nodes used, using more nodes on the second attempt may allow the query to complete within the required time.

Catastrophic failure requires rebuilding the compendium as describe in Data Storage.

Issues:

- Byzantine failures are not handles. Where to do failure detection? How to handle N-1 failures.

Programming Interface

Processing Pipeline

See Function:Troilkatt/Pipeline.

Parallel Spell

See Function:Troilkatt/Parallel Spell.

Evaluation

<To be updated>

Result and good (Spell), parallel system in 5 seconds, but on how many machines (effeciency). Linear scalability.

Downloading: correctness. But user can select updates.

Space usage: is part of effeciency.

Node failures.

Compendium studies:

- Number of zero elements in Spell distance matrices.

Performance microbenchmarks:

- Hbase read latency for a row with 1500 4 byte values for different number of columns.
- Hbae read latency when varying row size.
- Time to create a lookup table from DAT files.
- Perturbation of searches when updating lookup table with new datasets.
- Read performance of "bined" lookup table when varying bin size
- Write performance for "bined" lookup table
- Storage overhead of "bined" lookup table
- Overhead of starting a MapReduce job

Storage an integration:

- Ratio of datasets that could not be integrated

Search query part:

- Correctness: make sure the results for a query is identical to sequential version on yeast data.

- Interactive performance:
  - Time from "click on search" until results are displayed.
  - Use queries described in Spell paper.
  - Scalability test with regards to number query genes.
  - Scalability test with regards to number of datasets.
  - Scalability test with regards to number of genes in genome.

Issues to check and verify

A list of things to check that may influence the design or implementation of the system:

- Does hadoop configuration allow running multiple jobs simultaneously? What if a job is very unbalanced.
- Hbase dose support concurrent reads and writes to a table but how well does it work? For how long will the table creation/update job block readers if at all?
- PCL file format.
- Hadoop project with similar latency goals as Spell search? Some kind of aggregation library perhaps?

Previous Work

The Google papers:

- Google File System [4] (http://portal.acm.org/ft_gateway.cfm?id=945450&type=pdf&coll=ACM&dl=ACM&CFID=68902937&CFTOKEN=32427682)
- MapReduce [5] (http://portal.acm.org/ft_gateway.cfm?id=1327492&type=pdf&coll=ACM&dl=ACM&CFID=68902937&CFTOKEN=32427682)
- BigTable [6] (http://portal.acm.org/ft_gateway.cfm?id=1365816&type=pdf&coll=ACM&dl=ACM&CFID=68902937&CFTOKEN=32427682)
- Chubby Lock Service [7] (http://portal.acm.org/ft_gateway.cfm?id=1298487&type=pdf&coll=ACM&dl=ACM&CFID=68902937&CFTOKEN=32427682)

Online aggregation:

- Online MapReduce (query streams and online aggregation) [8] (http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-136.html)
- Performance and interfaces for distributed aggregation [9] (http://portal.acm.org/ft_gateway.cfm?id=1629600&type=pdf&coll=ACM&dl=ACM&CFID=68902937&CFTOKEN=32427682)
- MapReduce optimization [10] (http://portal.acm.org/ft_gateway.cfm?id=1555384&type=pdf&coll=ACM&dl=ACM&CFID=68902937&CFTOKEN=32427682)
- Tera-sort (how to reduce MapReduce startup time) [11] (http://developer.yahoo.com/blogs/hadoop/Yahoo2009.pdf)

Distributed query processing (in a cloud):

- Ad-hoc queries: [12] (http://portal.acm.org/ft_gateway.cfm?id=1454204&type=pdf&coll=ACM&dl=ACM&CFID=68902937&CFTOKEN=32427682)

Various papers:

- MapReduce vs. RDBM [13] (http://portal.acm.org/ft_gateway.cfm?id=1559865&type=pdf&coll=ACM&dl=ACM&CFID=68902937&CFTOKEN=32427682)
- Good tutorial on Ajax: [14] (http://www.adaptivepath.com/ideas/essays/archives/000385.php)
- Google Web Toolkit: [15] (http://code.google.com/webtoolkit/overview.html)

Future Directions

Templates

Various templates that are useful for writing this document:

A table with 3 rows and 3 columns:

Wenli

wenli	wenli	wenli	wenli
wenli	wenli	wenli	wenli

Foo	Bar	Baz
1.1	1.2	1.3
2.1	2.2	2.3
3.1	3.2	3.2

A box with some code:

```
void foo() {
  a = 2 + 2;
}
```

MediaWiki user guide (http://www.deakin.edu.au/dso/student/guides/qg-mediawiki-userguide.html#document)

Media:Zinfo_human_yeast.pdf