# Performance Analysis of MapReduce Implementations on High Performance Homology Search

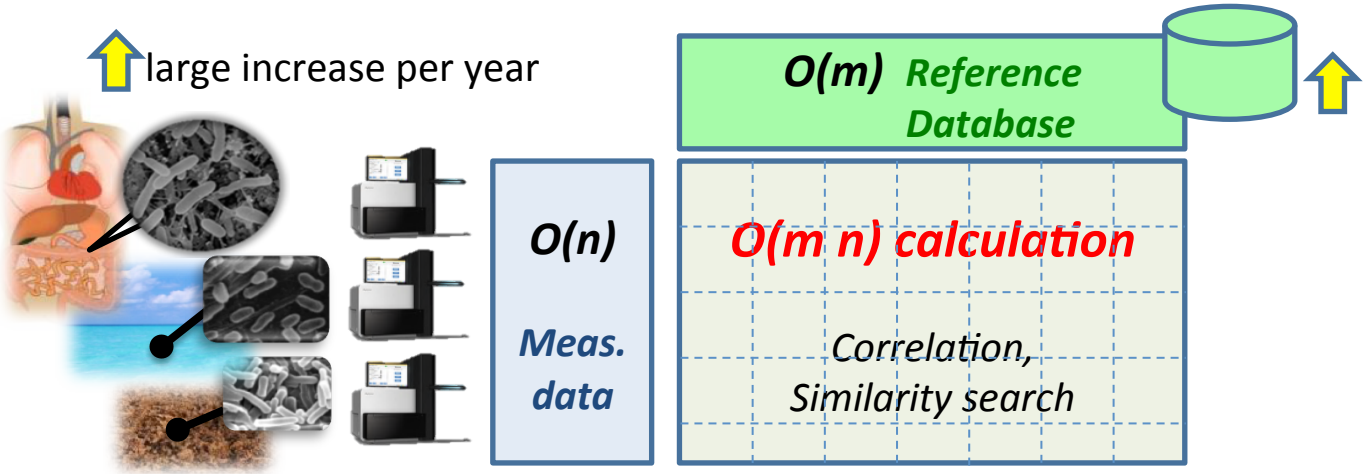
Chaojie Zhang, Koichi Shirahata,
Shuji Suzuki, Yutaka Akiyama,
Satoshi Matsuoka

Tokyo Institute of Technology

# Large-scale Metagenome Analysis

- Metagenome analysis using homology search
  - Directly sequencing uncultured microbiomes obtained from target environment, and analyzing the sequence data
  - Size of queries and database will reach GBs to TBs, and total compute data size will grow to product of the two (i.e. EBs to ZBs)

→ Utilize massively parallel data processing to cope with the increasing complexity



# Massively Parallel Data Processing on Homology Search

- Existing homology search algorithms
  - Sequential: BLAST [1], GHOSTX [2]
  - Distributed: mpiBLAST [3], GHOST-MP [4]

  → Parallelization is done with privately developed MPI-based master-worker frameworks
- MapReduce
  - Versatile big data programming model with associated software tool-chains
  - Conceal memory management in distributed systems

  → Apply MapReduce to large-scale homology search

- Problem: Unclear how to apply MapReduce to extremely large-scale homology search efficiently

[1] Altschul, S. F. et al.: Basic local alignment search tool, *Journal of molecular biology*, 1990
[2] Suzuki, S. et al.: GHOSTX: An improved sequence homology search algorithm using a query suffix array and a database suffix array, *PloS one,* 2014
[3] Darling, A. et al.: The design, implementation, and evaluation of mpiBLAST, *Proceedings of ClusterWorld*, 2003
[4] GHOST-MP webpage: http://www.bi.cs.titech.ac.jp/ghostmp/

# Goal, Approach and Contributions

- Goal
  - High performance MapReduce-based extremely large-scale homology search
- Approach
  - Implement MapReduce-based homology search
  - Performance analysis of MapReduce-based homology search
- Contributions
  - Describe MapReduce-based designs and implementations of a homology search algorithm
  - Preliminary experiments reveal that MapReduce exhibits good weak scaling and comparable performance with a MPI master-worker implementation

# Outline

- Introduction
- Background
- Proposal
- Experiment
- Conclusion

# Homology Search

- the method to search biological sequences similar to a biological query sequence in a database.
  - Mapping DNA sequence fragments to known protein sequences from public and private database
- BLAST: Basic Local Alignment Search Tool
  - Widely used homology search tool that uses a heuristic algorithm
- FASTA format
  - A sequence begins with a single-line description, followed by lines of sequence data

## Query/Database Input

```
>name 0|description 0
HPSKIPVIIERYKGEKQLPVLDKTKFL
VPD
>name 1|description 1
MKMRFFSSPCGKAAVDPADRCKEV
QQIRDQ
```

## Output

| Name | ... | E-value |
|---|---|---|
| hsa:124045... | ... | 2.04391e-76 |
| hsa:124045... | ... | 5.96068e-68 |
| hsa:124045... | ... | 1.38697e-32 |

# Improved Homology Search Algorithms: GHOSTX, GHOST-MP

- GHOSTX[1]
  - Adopts the BLAST's seed-extend alignment algorithm
  - Approximately 131-165 times faster than BLAST
    - Construct suffix array for query and database
    - Extend the seed till the matching score exceeds a given threshold for seed search
- GHOST-MP[2]
  - Extension of GHOSTX with MPI library for distributed computing environments
  - Utilize locality of database chunks

→ We use GHOSTX as a sequential implementation and extend it to MapReduce

[1] Suzuki, S. et al.: GHOSTX: An improved sequence homology search algorithm using a query suffix array and a database suffix array, *PloS one,* 2014
[2] GHOST-MP webpage: http://www.bi.cs.titech.ac.jp/ghostmp/

# Problems on MapReduce-based Homology Search

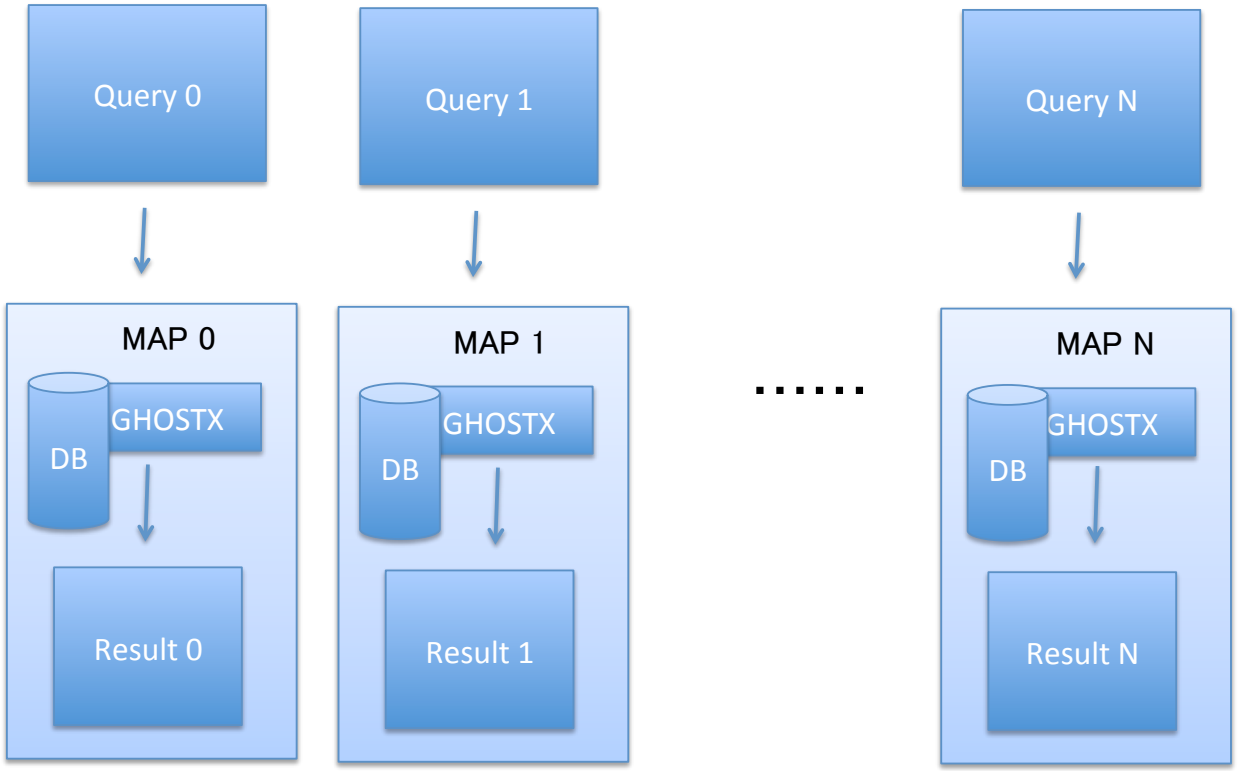
- How to design and implement homology search algorithms onto the MapReduce model
  - How to handle two different datasets, queries and database, using MapReduce
- Whether the massive resource required would overwhelm MapReduce

# MapReduce-based Designs of Homology Search

- Basic idea
  - distribute query data onto multiple Mappers
- We consider two different designs based on assignment of database onto multiple nodes
  - Database replication
  - Database distribution
  - Query data is distributed on both designs, while database allocation policies are different

# MapReduce–based Design with Database Replication

- Input query data files: copied to a distributed file system
- Database file: replicated onto local disk on each compute node
- No reduce stage
- Useful when database is small



# Implementations of MapReduce

- Hadoop[1]
  - Implemented in Java
  - Consisted of Hadoop Common, Hadoop Distributed File System(HDFS), Hadoop YARN, Hadoop MapReduce
- Spark[2]
  - Built on top of HDFS
  - Resilient distributed dataset(RDD)
- Hamar[3]
  - Scalable MPI-based MapReduce with hierarchical memory management
  - Run on either CPUs or GPUs

→ We implement the database replication design on these MapReduce implementations

[1] White, T.: "Hadoop: the definitive guide: the definitive guide." O'Reilly Media, Inc., 2009
[2] Zaharia, M. et al.: "Spark: cluster computing with working sets." 2nd USENIX conference on Hot topics in cloud computing. 2010
[3]Shirahata K. et al.: "Out-of-core GPU Memory Management for MapReduce-based Large-scale Graph Processing", IEEE Cluster 2014.

# Implementation on Hadoop

- Database Replication Design
  - Use Hadoop Pipes to use GHOSTX on top of Hadoop
  - Pass queries as input
    - Implement WholeFileInputFormat to avoid splitting
  - Read database from local disk directly

```
hadoop pipes\
  -D hadoop.pipes.java.recordreader=true\
  -D hadoop.pipes.java.recordwriter=true\
  -files [db_files]\
  -input [input_dir]\
  -output [output_dir]\
  -inputformat WholeFileInputFormat\
  -program ghostmr
```

# Implementation on Spark

- Database Replication Design
  - Use RDD pipe() operation to call GHOSTX binary from Spark
  - Read query from HDFS as input
    - use jar file including WholeFileInputFormat
  - Read database from local disk directly

```
spark-submit\
  --class "GhostMR"\
  --master yarn-client\
  --num-executors [num_nodes]\
  --executor-cores [num_threads]\
  --files [db_files]\
  --jars lib/hadoop-mapreduce-client-core-[ver].jar\
  ghostmr.jar
```

# Implementation on Hamar

- Database Replication Design
  - Call GHOSTX directly from map function
  - Read both query and database from local disk
  - Assign the same amount of query data to each node
  - Execution
    - mpirun -n np -hostfile hosts ghostmr tablefile
  - tablefile
    - Tab-separated query, database, and output file names

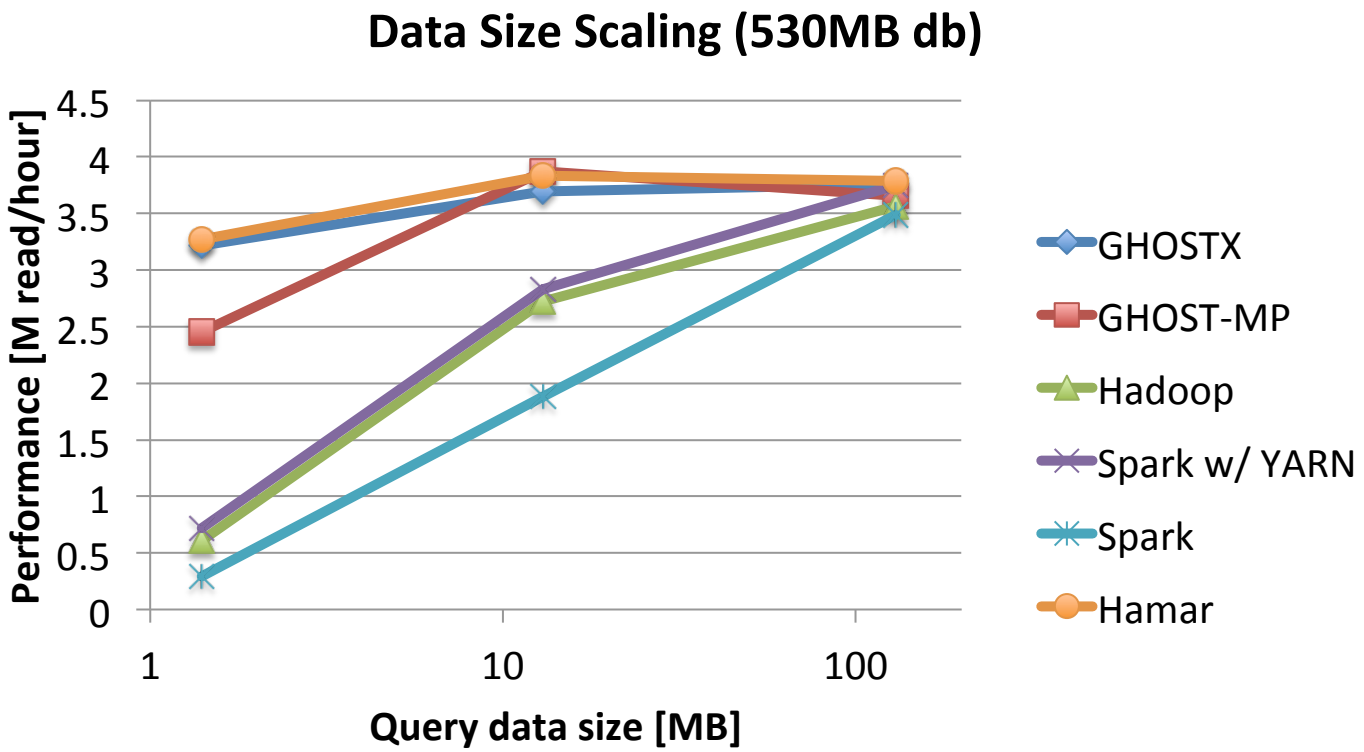
# Experimental Setup

- Compare performance with different size datasets
  - understand performance characteristics of MapReduce implementations
- Data Resource
  - FASTA database, “nr”, The National Center for Biotechnology Information website
    - we split the database into different sizes for various size comparison
  - FASTA query, “SRS014107”, Data Analysis and Coordination Center for Human Microbiome Project website
    - we split input query files into 10MB of smaller files before putting them to HDFS for Hadoop and Spark
- Machine Configuration
- Software Version
  - Hadoop 2.4.1
  - Spark 1.1.0

| CPU | Intel® Core™ i7-3930K (12 cores, 3.20GHz) |
|---|---|
| Memory | 48GB |
| Local SSD | 102GB |
| Compiler | GCC 4.4.5 |
| InfiniBand | FDR 4X (56Gbit/s) |

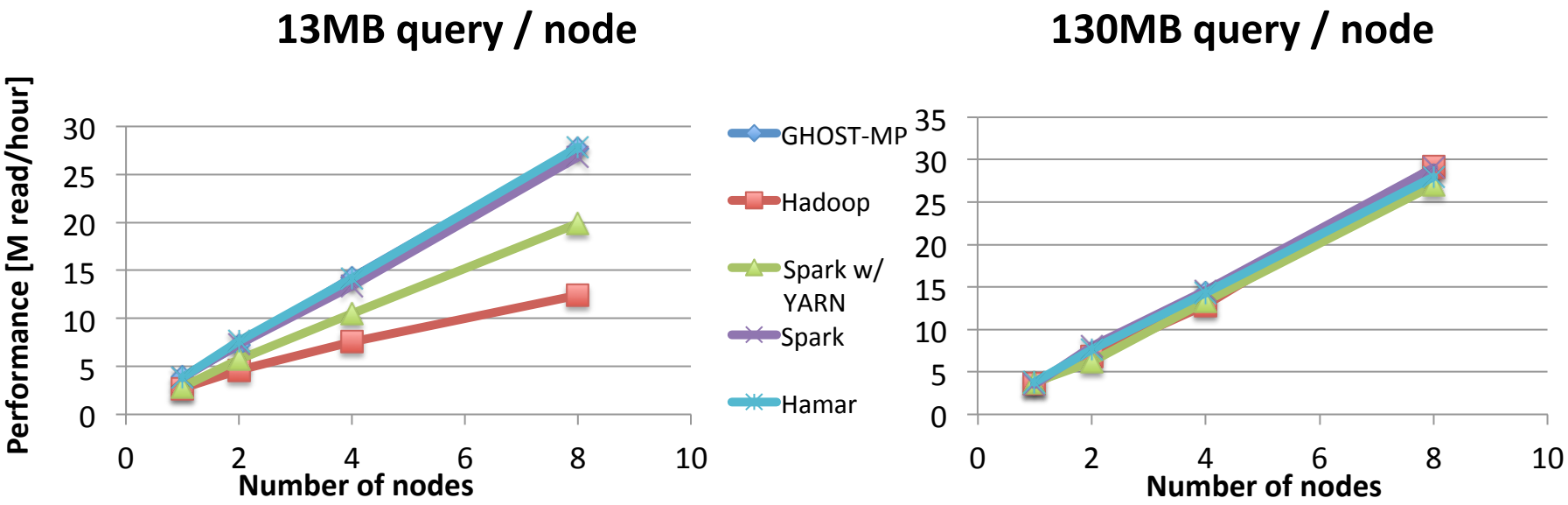
# Query Data Size Scaling

- Use single node with 530MB database
  - we split input query files into 10MB of smaller files before putting them to HDFS for Hadoop and Spark
- Similar performance when query size gets larger on single node



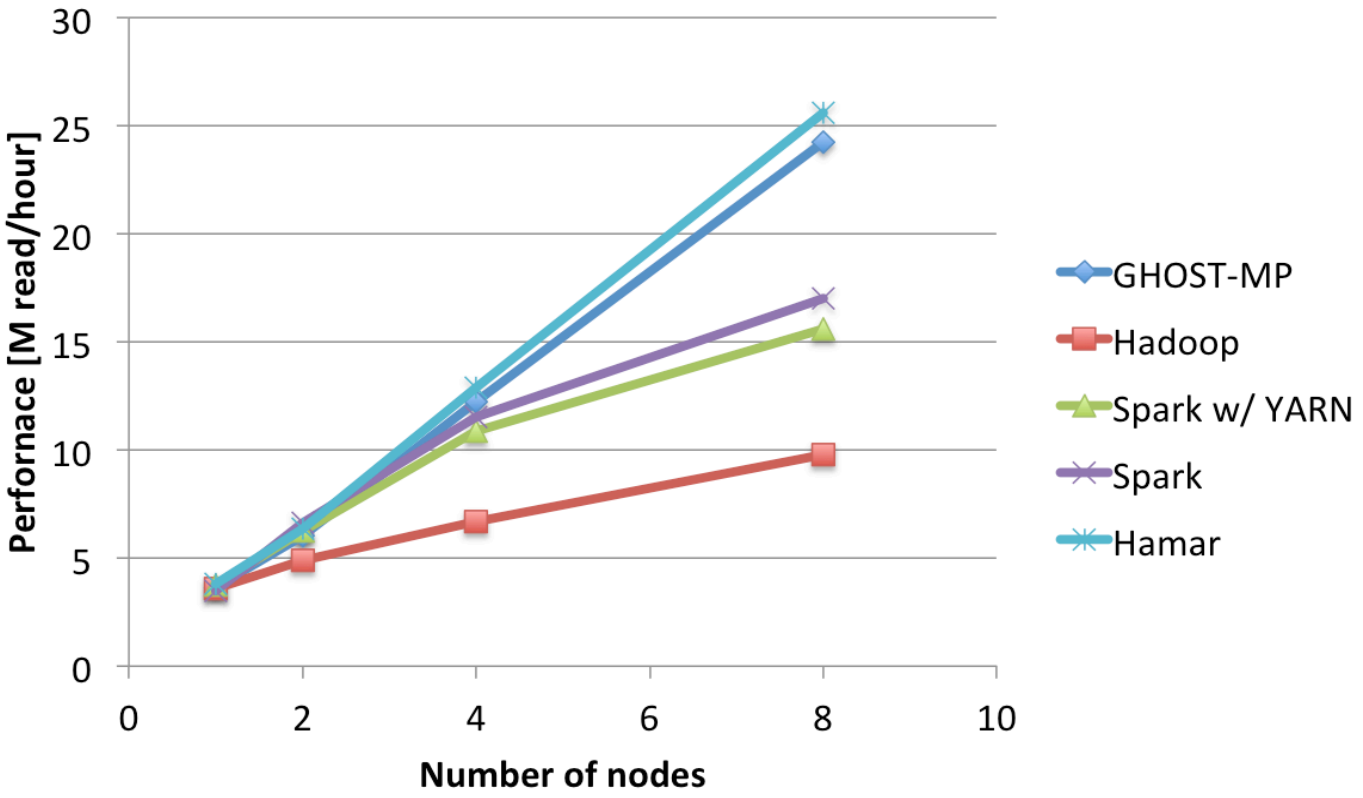
# Weak Scaling Performance

- Use multiple nodes with 530MB database, and 130MB query per node
  - we split input query files into 10MB of smaller files before putting them to HDFS for Hadoop and Spark
- All the implementations exhibit good scalability for large query



# Strong Scaling Performance

- Use multiple nodes with 530MB database, and 130MB query in total
  - we split input query files into 10MB of smaller files before putting them to HDFS for Hadoop and Spark
- Overhead becomes more obvious when number of nodes gets larger



# Related Work

- K MapReduce (KMR)[1]
  - Optimizes shuffle operation utilizing interconnect on K computer
  - Conduct experiments using GHOST-MP by replacing master-worker tasking library
  - did not compare with other existing MapReduce implementation
- mpiBLAST[2]
  - Applies database segmentation
    - Each node searches a unique distributed portion of database
  - High optimized only for BLAST

[1] Matsuda, M. et al.: K MapReduce: A scalable tool for data-processing and search/ensemble applications on large-scale supercomputers, IEEE Cluster, 2013
[2] Darling, A. et al.: The design, implementation, and evaluation of mpiBLAST, *Proceedings of ClusterWorld*, 2003

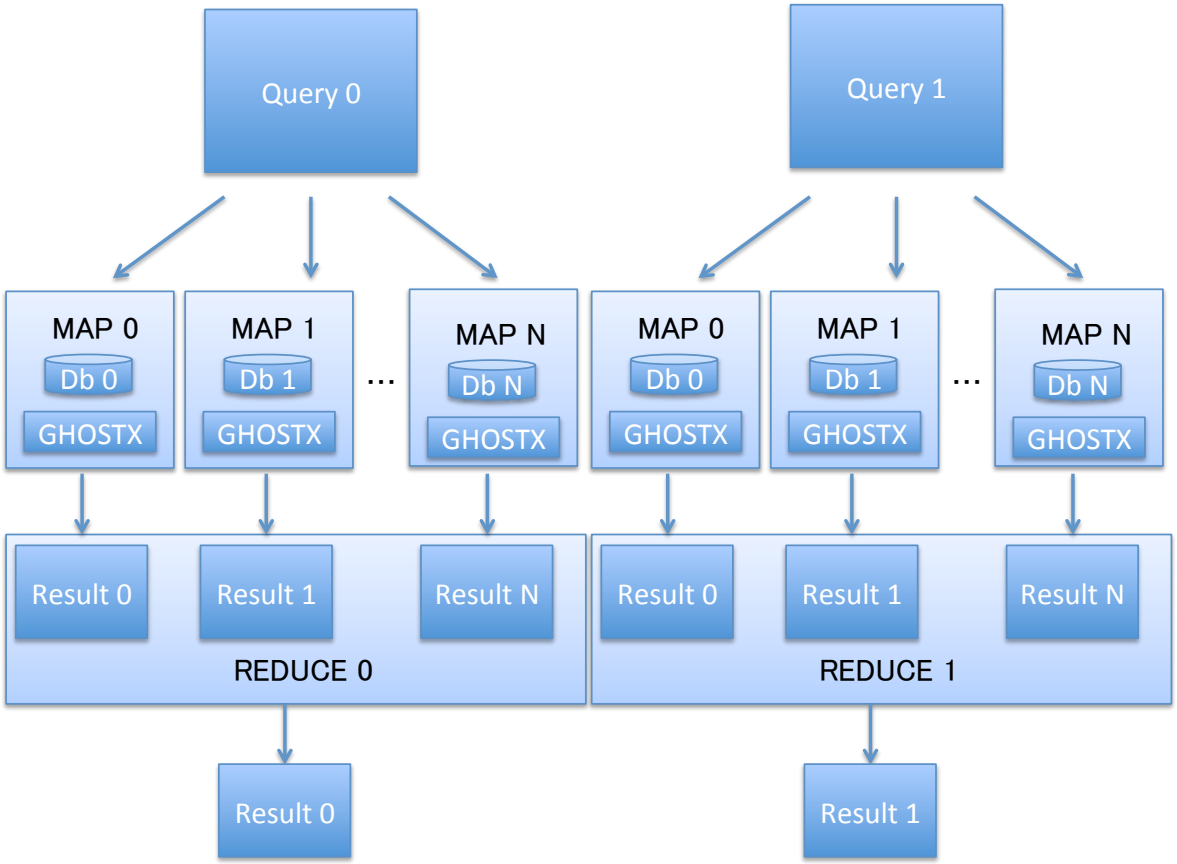
# Conclusion

- Conclusion
  - We proposed MapReduce-based Designs and Implementations
  - MapReduce implementations exhibit good weak scaling and comparable performance with GHOST-MP
- Future work
  - Implement other MapReduce-based homology search designs
  - Conduct further detailed performance analysis
    - Larger dataset
    - Large-scale computing environments
    - Further breakdown analysis

- Backup

# MapReduce–based Design with Database Distribution

- Input query data files: copied to a distributed file system
- Database file: split to multiple chunks and each chunk distributed on each node



# Database Size Scaling

- Use single node of Raccoon with 1.3MB query