Comparison of Distributed Data-Parallelization Patterns for Big Data Analysis: A Bioinformatics Case Study

Jianwu Wang, Daniel Crawl, Ilkay Altintas

San Diego Supercomputer Center, University of California, San Diego

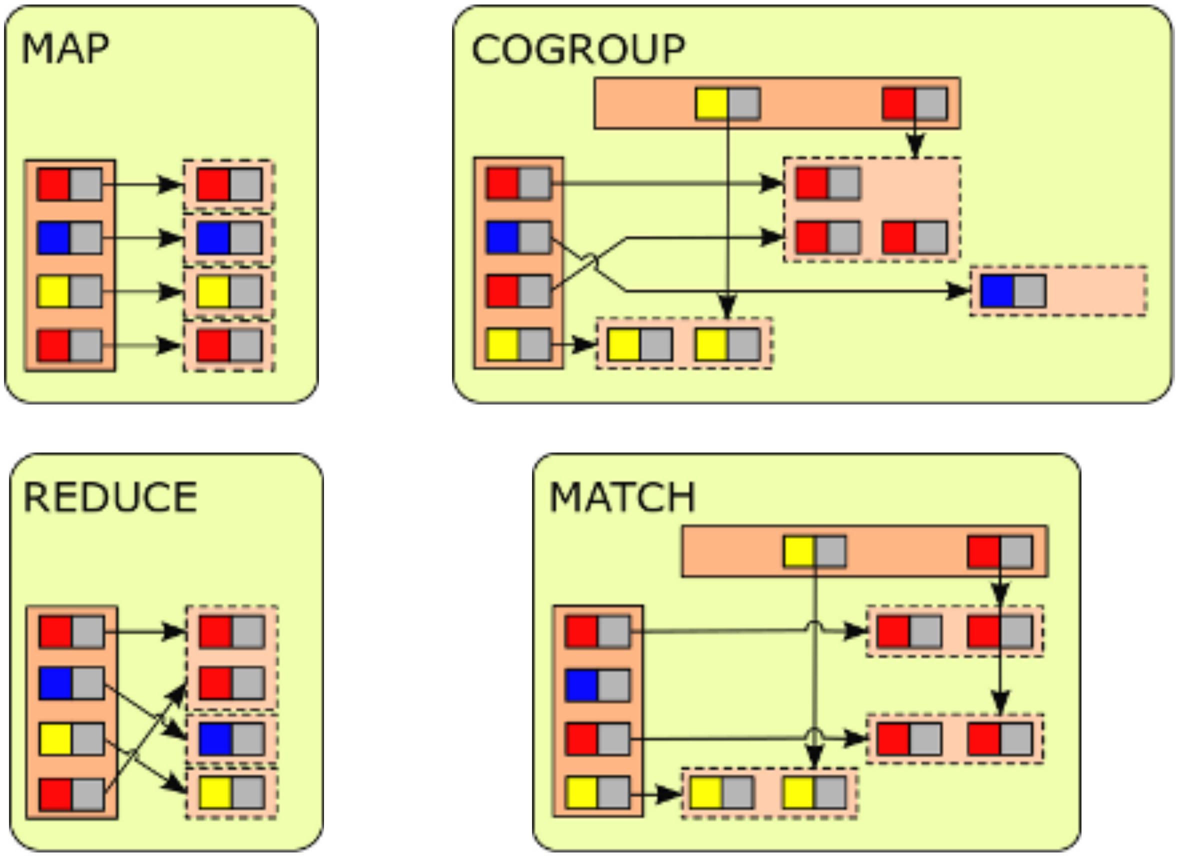
Kostas Tzoumas, Volker Markl

Technische Universität Berlin

Background – DDP Patterns

- **Distributed Data-Parallelization (DDP) Patterns**
  - Many identified DDP Patterns: Map, Reduce, Match, CoGroup, and Cross (a.k.a. All-Pairs).
  - Reusable practices for efficient design and execution of big data analysis and analytics applications.
  - Combine data partition, parallel computing and distributed computing technologies.

Background – Details on Some Identified DDP Patterns



Challenges

- **Which DDP patterns fit for a specific program? Which one is the best? What are the main factors affecting the performance?**
  – Comparisons of different DDP patterns on performance when applied to the same tool and the main factors affecting such performance have not been well studied.

Work Summary

- Using an existing bioinformatics tool as an example, called CloudBurst, demonstrate multiple feasible DDP options for the same tool.
- Identify two key factors affecting the performances of different DDP options.
- Demonstrate the feasibility of the identified factors and show that switching DDP option could speed up performance by over 1.8 times.

A Bioinformatics Case Study - Background

- **Sequence Mapping Tools**
  - Map query sequences to reference sequences to know whether there are similar fragments in reference data for each query sequence and their locations.
- **Seed-and-Extend Algorithm for Sequence Mapping**
  - First finds sub-strings called *seeds* with exact matches in both query and reference sequences;
  - Then extends the seeds into longer, inexact matches.

A Bioinformatics Case Study - CloudBurst

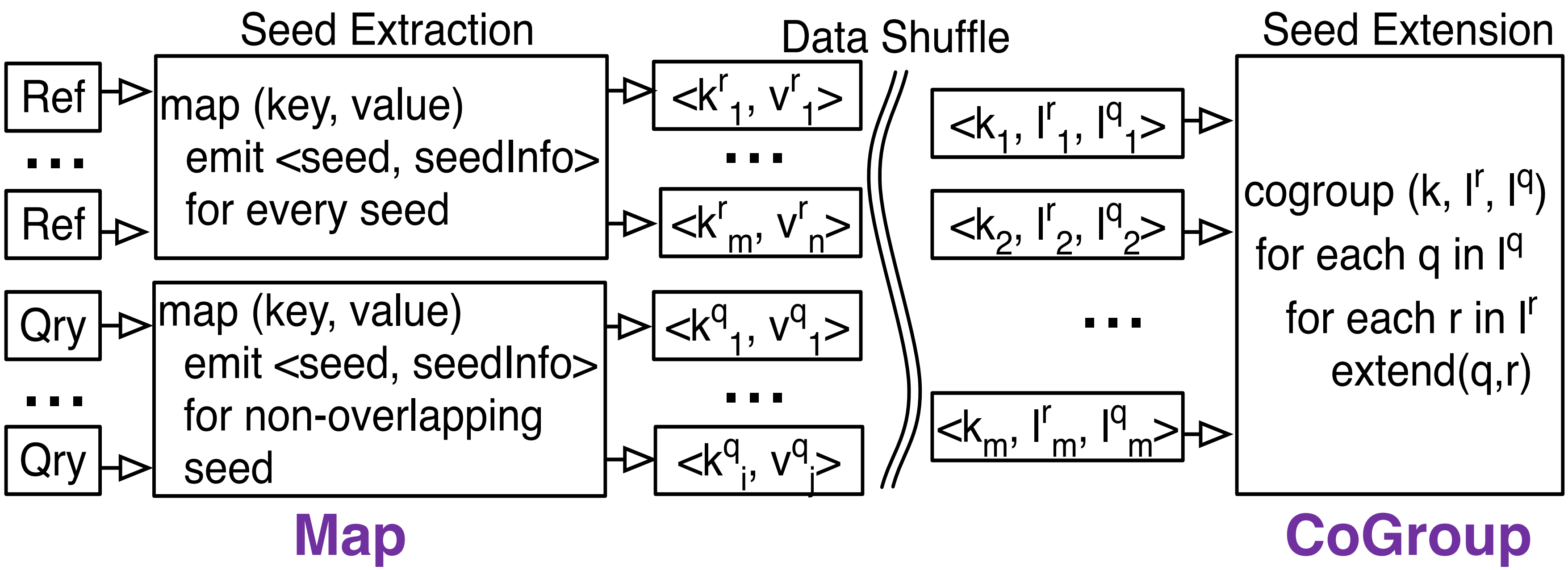
- **CloudBurst : a Parallel Seed-and-Extend Sequence Mapping Tool**
  – Its scalability and performance speedup in distributed environments have been verified.
  – Its original implementation is based on MapReduce.
  – We re-implemented it using MapReduce, MapCoGroup, MapMatch.

Original CloudBurst using MapReduce

Map **Reduce**

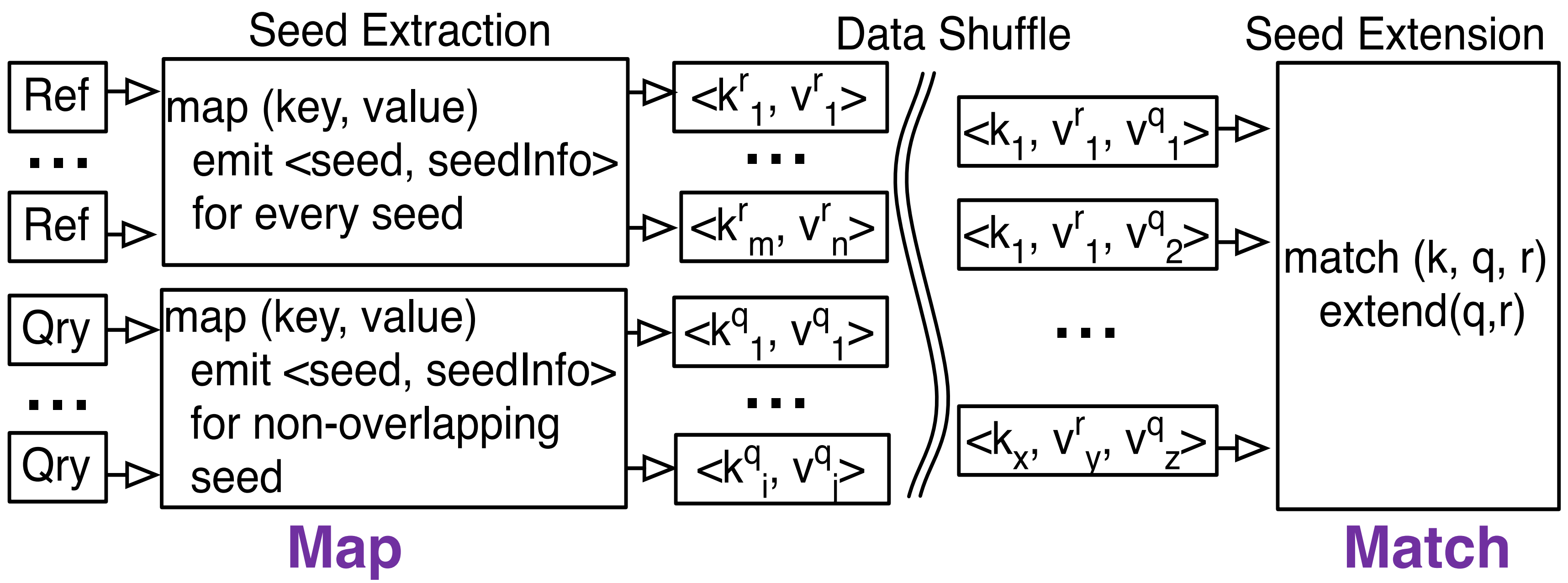
Query and reference datasets of CloudBurst have to be distinguished throughout the phases.

CloudBurst using MapCoGroup



Map: Two Map functions for query and reference separately.
CoGroup: Each instance gets a reference list and a query list for the same key, so it has one less loop compared to CloudBurst using MapReduce.

CloudBurst using MapMatch



Map: Two Map functions for query and reference separately.
Match: Each instance gets one query value and one reference value for the same key, it does not need any of the loops in CloudBurst using MapReduce.

DDP Performance Comparison

- **Main difference of the above three implementations**
  - How the Map output data is read into and processed in Reduce/CoGroup/Match?
- **Total execution time of Reduce/ CoGroup/Match includes two main parts**
  - User function execution time
  - User function execution number

The First Performance Factor

- **The difference between the numbers of keys in the two input data, denoted as *p*.**
  - It reflects the balance of the two input datasets.
  - If one dataset is much larger than the other one, their key sets will have less common keys.
  - If a key only exists in one dataset,
    - Match will not have user function executions for it.
    - Reduce still needs to run user function executions for it.
  - Reduce is more suitable for balanced input datasets and Match is more suitable for imbalanced ones.

The Second Performance Factor

- **The average number of values per query/ reference key, denoted as *q*.**
  - It reflects the sparseness of the values for each key.
  - If a key has a lot of values,
    - Match has to have a separate user function instance for each possible value pairs.
    - Reduce only needs one execution to process all values for the same key.
    - Reduce has less user function execution number.
  - Reduce is more suitable for condensed values per key and Match is more suitable for sparse values per key.

Performance Analysis for CoGroup

- **User function execution time**
  - CoGroup takes less time than Reduce since it does not need the first loop in Seed Extension phase.
  - It takes more time than Match because it have unnecessary executions with empty set from one input.
- **User function execution number**
  - This number for CoGroup is the same with Reduce's and less than Match's.
- **Overall, its total execution time should be between those of Reduce and Match.**

Questions to be Answered by Experiments

- Would changing the DDP pattern to execute the same function have a big influence on the performance?
- Can the two factors identified above adequately explain the performance differences?

Experiment Information (1)

- **DDP Execution Engine**
  - We use Stratosphere (version 0.2) because it supports Map, Reduce, CoGroup and Match directly.
- **Test Bed**
  - It is done on five compute nodes in a compute Cluster environment.
  - Each node has two four-core CPUs.
  - We only run the programs with a static environment because the target is to compare performance differences of the DDP patterns, not scalability.

Experiment Information (2)

- **Execution Parameter for CloudBurst**
  - *mismatches* (*k*) specifies the maximum allowed length of differences. It affects the results greatly.
  - Both values of *p* and *q* will change accordingly when *k* value changes.
  - So we tested different executions of the same program and parameters except *k* value.
- **Parallelization Parameter**
  - All experiments are done with 12 parallel instances for Map, Reduce, Match and CoGroup.

Experiment Information (2)

- **Experimental Data**
  - The first experiment processes two large datasets from real projects.
    - Query dataset: over nine million sequences.
    - Reference dataset: over 1.2 million sequences.
  - The second experiment processes only a large reference dataset.
    - Query dataset: only include the first 5000 sequences used above.
    - Reference dataset: the same as above.

Experiment Results for Execution Times (1)

The execution times (unit: minute) of different DDP implementations of CloudBurst for large query and reference.

Mismatch number (*k*)	0	1	2	3
MapReduce	2.786	3.405	3.537	8.622
MapCoGroup	1.564	1.916	2.477	24.640
MapMatch	1.474	1.883	2.689	47.393

Finding:

1. The performances of MapMatch is better than those of MapReduce for $k = 0, 1, 2$; but much worse when $k = 3$.
2. MapCoGroup's execution times are always between those of MapReduce and MapMatch.

Experiment Results for Execution Times (2)

The execution times (unit: minute) of different DDP implementations of CloudBurst for only large reference.

Mismatch number (*k*)	**0**	**1**	**2**	**3**
MapReduce	1.920	2.313	2.565	2.538
MapCoGroup	1.523	1.754	1.907	1.888
MapMatch	1.453	1.690	1.763	1.799

Finding:

1. The performances of MapMatch are always better than those of MapReduce for this experiment.
2. MapCoGroup's execution times are always between those of MapReduce and MapMatch.

Finding Summary

- Different DDP patterns have great impact on the execution performance of CloudBurst.
- No DDP pattern combination is always the best, even only for different parameter values of the same tool.
- DDP pattern selection of the same tool could be very important for its performance.

Experiment Results for Factors

Relationship between execution speedup and its factors for large query and reference.

Mismatch number (*k*)	**0**	**1**	**2**	**3**
Key set size difference (*p*) (unit: million)	167	163	116	0.28
Average value number per key (*q*)	1.6E-5	1.6E-2	6.05E-1	2.73E3
Speedup ratio of MapMatch to MapReduce	1.890	1.704	1.201	0.181

Relationship between execution speedup and its factors for only large reference.

Mismatch number (*k*)	**0**	**1**	**2**	**3**
Key set size difference (*p*) (unit: million)	179	189	149	4.16
Average value number per key (*q*)	0	1.8E-5	4.6E-4	1.74
Speedup ratio of MapMatch to MapReduce	1.321	1.369	1.455	1.411

Finding Summary

- The values of the two factors greatly affect which DDP pattern has better performance.
- Most speedup ratios decrease along with the decrease of *p* values and the increase of *q* values.

Conclusions

- Different DDP patterns could have a great impact on the performances of the same tool.
- MapReduce can be used for wider range of applications with either one or two input datasets. But it is not always the best choice for application complexity and performance.
- Two affecting factors, namely input data balancing and value sparseness, can explain their performance differences.

Future Work

- Investigate more tools for multiple DDP patterns and their performances on other DDP engines to generalize our findings.
- Study how to utilize the identified factors to automatically select the best DDP pattern combination from multiple available ones.

- **Acknowledgements**
  - NSF ABI Award DBI-1062565 for bioKepler
  - The Gordon and Betty Moore Foundation award to Calit2 at UCSD for CAMERA
  - The rest of bioKepler and Stratosphere teams for their collaboration
  - FutureGrid project for experiment environment support
- **Contact**
  - jianwu@sdsc.edu