Summary: All the adjustment in the paper in response to the comments are highlighted inside the paper in blue color.

Confluence: Improving Iterative Distributed Operations by

Key-Dependency-Aware Partitioning

**Comment 1:** "Confluence constructs a key dependency graph based on the application semantics to reflect the key dependency relationship of the

datasets of adjacent computing iterations"

also:

"In the key dependency graph proposed in this paper, the

construction of the graph does not use any data partitioning

information, but considers the \*\*application semantics\*\* of the

keys of the data, which depends on the application logic

only.

\*\*  You better define what is "application semantics" earlier.

    Remember I also raised the issue in your thesis review "

"What do you mean “semantics” of the distributed operations? Sorry, I still don't get it.”

Response: I have defined the “application semantics” in the fourth paragraph of the Introduction Section: “The application semantics refers to what new key-value data entries a key-value data entry will generate in a distributed operation, based on the logic of the program.”

But to make it more clear, I now update it to:

“The application semantics

of iterative distributed operations, also called distributed

operation semantics in this paper (Section 3.1), reflects the

logic of the program that decides, given a key-value data

entry as the input, what new key-value data entries will be

generated as the output in every iteration of the distributed

operation. The key dependency relationship between diffe-

rent computing iterations are derived from the distributed

operation semantics. It represents the transformation pro-

cess from key-value pairs to new key-value pairs in iterative

distributed operations and it can be expressed to computer

by means of the Key Dependency Graph (Section 3.2). It

is special to every iterative distributed operation and is

decided by the logic of the code and the input dataset.”

-----------------------------------------------------------------------------------

**Comment 2:** "In fact, we usually do not know the contents of Ai and

cannot explore the partition scheme for the next iteration

until the program has actually finished the ith iteration.

Due to this limitation, the idea of finding out the optimal

partition schemes for all the iterations to minimize the

overall data transfer size S is infeasible."

But in your Algorithm 1:

Input : Key Set Ki, containing all the possible keys

(or key patterns) of the output key-value

pairs of any Iteration i;

1. What do you mean "all possible keys" of the output key-value pairs?

2. It seems to me you have to collect ALL the keys from all output key-value pairs.

    Then, reconstruct the Key Dependency Graph. Gathering ALL these information

     from all nodes could be a bottleneck, even you just collect the keys.

3. Why there are "unknown keys" (I guess this is about the “semantics” of the distributed

operations?

Response:

1. “all the possible keys”means all the key values. But I also say that it can be all the possible “key patterns”. When the number of key values is large, the number of key pattern can be very small. I say all the keys here because I do not want to lose the generality of the KDG construction algorithm. I see your wonder. To clarify, I added:

“In practice, the input set K i in Algorithm 1 is

defined to contain the key patterns instead of the distinguished key values, so that a node in the KDG represents

a key pattern. Such practice limits the width of the KDG in

each level and our algorithm introduced later can partition

the key-value pairs based on their key pattern. In the rest

of this paper, unless specified, the input set K i is the key

pattern set, but for generality, we still refer to it as the key

set.”

I also adjust the definition of the key dependency to include the key pattern.

I think it also answers your second question.

3. The unknown key dependency is only raised when there are some key values (or key patterns) the programmer cannot make sure in a computing iteration.

----------------------------------------------------

**Comments 3:** CONFLUENCE

Even though it is hard to find the optimal partition solution

to minimize the overall shuffle size, if the application

semantics of the specific distributed operations (distributed

operation semantics) is already known, the shuffle size can

be reduced to the maximum extent by exploring the key

dependency of different iterations.

Q1: How to obtain "distributed operation semantics"? Reading the Spark code?

\*\* You see all the troubles are from “semantics” of the distributed operations

\*\* You may like to add a special section just to talk about this.

Response: Yes, by reading (usually, when writing) the code.

I add a section (Section 3.1) to explain the semantics issue:

“3.1 Distributed Operation Semantics

Given a key-value pair as the input to an iteration of the

distributed operation, it will generate a (or a set of) new

key-value pair(s) based on the logic of the program. The

distributed operation semantics represents such logic of the

generation of key-value pairs from the input key-value pairs

in every iteration of the distributed operation. By saying

that we know the distributed operation semantics for a

specific application, it means that we know what the output

key-value pairs will be for an input key-value pair in a

computing iteration. The programmer is supposed to be well

acquainted with the distributed operation semantics.

We use the key dependency graph (introduced later) to portrait

the distributed operation semantics so that

it is comprehensible to the computer. Every

distributed operation has its own semantics and it relies

on the programmer to understand the distributed operation

semantics and construct the key dependency graph.”

-------------------------------------

**Comments 4:** 4.2 MultiAdjacentList

"However, the iterative operations can be divided into several smaller

iterative operations by dividing the dataset as in Fig. 4(d). In

the iterative operations in the dashed boxes, the pure confluence

subgraphs exist in their KDG’s by the key dependency Li ) (Li; tails)."

Q1: Who can tell "the iterative operations can be divided into several smaller

iterative operations by dividing the dataset as in Fig. 4(d)" -- Your algorithm or by human?

Response: By human. Not the algorithm. The logic of the algorithm can be unchanged.

Similar problem in 4.1 MovieLensALS

"However, if the iterative distributed operation is regarded

as two individual iterative operations and is divided

as in Fig. 4(b), the pure confluence subgraph can be found"

\*\* Who knows (or can detect "the iterative distributed operation is regarded

as two individual iterative operations and is divided as in Fig. 4(b)"?

Response: The human (the programmer).

Maximizing Shuffle Network Bandwidth Utilization by Application-Level Flow Scheduling

\*\* The key work "Application-level" (if important) was not emphaized in Abstract.

\*\* The "application-level scheduling" was mentioned briefly in Introduction:

"The application-level scheduling approach works by making use of both the shufflerelated information"

  But it is unclear to me what exactly  is the application-level approach.

  Please note that the key design of Yarn is to enable multiple applications (MapReduce, Spark, MPI)

  runing on the same cluster at the same time. I am not sure if those shuffle related information

  is about a sigle application (e.g., one MapReduce program), or among all applications (e.g., from Resource

  Manager's view, it knows the status and resource usage for all applicaions running on it -- to me, these are all

  "application-level", too).   So, you better make it clearer.

\*\* Concern: if we have multiple MapReduce programs running at the same time, is your "application-level scheduling"

    applied on individual program separately? Then, without the global information, can you achieve optimal results

    for all 3 programs? In this regard, it seems the network-level approach is better (enable global optimization).

   (I will keep reading the rest of contents and see what you are actually doing -- Later I found your scheduler is running within

    Resource Manager. This seems reasonable.).

\*\* "network is a heterogeneous": usually we won't consider this is a mainstream problem as most clusters are made

    homogeneously (just my own view). We would focus more on the dynamic network traffic in real time computing

    (that is , how to do proper shuffle given the available network bandwidth at run time?)

    I feel you are just creating problems to solve.

\*\* "The simulation results show that comparing to the random-source-selection method adopted by

     YARN, PGSS can improve the cluster network bandwidth utilization especially in a heterogeneous .."

   ---> Just comparing with a dump solution (random-source-selection) can not show your work is better

     than those state-of-the-art solutions ===> If  I am a reviewer, I will use this reason to kill your paper ^\_^

\*\* "Minimizing the completion time of the coflow is NOT the

same thing as minimizing the shuffle completion time."

Be reminded, you have not define "shuffle completion time" earlier.

The readers may not get what you mean, and why this was emphasized.

\*\* "To obtain the optimal scheduling solution that minimizes the shuffle completion time,"

  I believe we have raised the question during your oral exam: if you just optimize shuffle completion time

  will you sacrify the total exacution time. I believe the key objective is to minimize the total execution time, right?

  Please add your justification ! Is it only valid if the application is

  Again, "shuffle completion time" was not defined. So I won't be able to

  give further comments. Surely, if you only refer to "shuffle-heavy" MapReduce tasks,

  reducing shuffle time can help to improve the total time (since the shuffle time dominate the total execution time).

3.1.1 Design Considerations

\*\* What are the actual problems behind the scene?

I found you keep discussing the the number of fetchers and the communication pattern

of the fetch flows. Are there other factors, like (1) nodes are busy with the map tasks, thus can not help to distribute data;

 (2) reducers were activated too early thus occupy resources but not receiving data?

----------------------------

Minor:

Gideon-II cluster [33] having a heterogeneous network ??