Spark-BDD : A Big Data Debugger

Muhammad Ali Gulzar
UCLA

Tianyi Zhang
UCLA

Seunghyun Yoo
UCLA

ABSTRACT

Apache Spark has become a key platform for Big Data Analytics, yet it lacks complete support for debugging analytics programs. As a result, the development of a new analytical toolkit can be a painstakingly long process. To fill this gap, we developed Spark-BDD (Big Data Debugger), which brings a traditional interactive debugger experience to the Spark platform. Over the course of this project we integrated current version of Spark-BDD with some interactive fault localization and optimization mechanisms. Analytic programmers (e.g., data scientists) can leverage Spark-BDD interactive debugging capabilities to set breakpoints and watchpoints, investigate crashes and failures, identifying straggling data partitions. Along with debugging this tool provides some useful runtime information that can be helpful in improving the performance of spark in an unpredicted cluster deployment. One of these optimizations is inter-stage repartitioning to avoid data skew during data shuffling where performance improvement is reflected in the spark job competition time. Most of these debugging features pose minimum overhead and in turn provide valuable debugging with lowest number of iterations.

Keywords

Debugging, Apache Spark, Fault Recovery & Localization, Performance

1. INTRODUCTION

Large scale data processing is the hottest topic these days. As the accessibility of the technology and methods to collect is improved , the amount of data is also increasing exponentially. Processing these massive amount of datasets are both difficult and necessary. As scene from he experience single computer super machines are capable enough to to analyze that scale of data. To overcome that cluster computing frameworks are the right path to perform these kind of analysis. Large companies like Google and Facebook have there own data center which they use to process their data with small companies lease subset of inter connected machines from third party resource provider like amazon. Since the availability of these kind infrastructure is not an issue anymore , the need for software framework that be ran on it is imminent. Apache Spark is one of these large scale data processing framework that can be run cluster computing frameworks with thousands of nodes. Spark is built on MapReduce programming paradigm from LISP but also provides data processing operation like GroupBy and Joins. Several other frameworks before Sparks also encountered the same problem in the sam distributed way but most of them were either slow (Hadoop) or complicated (Dryad). Spark cater both of these issued by providing an in-memory model for lighting fast speed and simple Scala, java and python API for easier understanding. The choice of Spark for this project is because of its growing popularity in the community, impressive performance and research support as a lot go research lab are working in or on top of of spark.

In this project we present a debugging tool called SparkBDD build on top of Spark to debugging feature that are usually available to user in single process programs. The basic aim was to develop a tool corresponds to the features available to users on conventional debuggers as the users of Spark are mostly ML and Data Science that are are not very familiar with the spark internals. By hiding the internal details, SparkBDD intruduces a layer of debugging build in the same API and a web based UI to perform debugging both physical and data layer. We support features like breakpoint , Watchpoint, stragglers profiling, crash culprit identification and recovery. All of these features supports wide array of use cases and drives a new optimizations and performance improvements from the measurements collected during debugging runtime. Later in the report we also show the overhead of this frameworks and one of the performance improvements based the stragglers profiling information.

2. MOTIVATION

With the abrupt increase in the adaptation of these frameworks, debugging these frameworks are getting challenging too. Current state of these frameworks does not allow any kind of debugging support. The only debugging support that comes build in with analytic frameworks are single process mode where user can run a job in single computer with subset of data. But still this does not add any thing to the fault localization task as it lacks the distributed infrastructure, actual dataset and resource sharing. In case of Spark, the application written by user does not comply with the actual execution because lazy evaluation of RDDs and some optimizations. Running a step by step debugger of Scala and Java like JDB, would not help just like a multithreaded applications. Since everything is distributed , the actual tasks are executing at the works which are hard to track from current logging mechanism. Spark logging can help at the physical layer of the framework where you can have abstract information of the tasks while debugging at data layer is necessary.

Cluster competing frameworks are usually resource hungry and also consumes a lot time to execute a job. Once the

job is submitted the user is kept in the dark until the end of the job or the occurrence of the failure. In case of failure all the previous computations are wasted because of spark in memory design and the user need to start the job from start by doing the redundant work. This scenario resulted in both resource and time wastage and the later the failure occur more wastage would happen (If not cached). Early detection and identification is crucial for effective debugging and resource saving. All these drawbacks of using analytic frameworks motivate the development of a debugging environment to can enable users detect the logical errors in their implementations and refine the source of the fault on the data plane.

3. APPROACH

Considering the motivations and the current debugging need of Spark, we built a debugging environment integrated within spark to support most of the features to performa extensive debugging both on physical and data plane. Spark-BDD is build with in the internals/core of Spark to have control over the low level details of then jobs so that even the minute actions could be shown to the user as per need. This debugger also provides an interactive UI designed along the with the current Spark monitoring UI to give low level controls to the user. The idea here is to provide essential details to the user to detect faults and logical errors that can not be detected through the normal logging mechanisms. Interactive user interface provides runtime details to the user to make debugging decisions like step in/out, put monitors, handling crashes, tracing faulty data source and tracking latencies.

4. RELATED WORK

TODO

5. IMPLEMENTATION - FEATURES

The current state of the debugger performs the features described in the sub sections below. All the features are implemented with in the internal core code of spark as hooks to provide the most efficient and robust implementation. Along with the debugging backend systems , almost all the features are supported and controlled through both programming API in both Java and Scala and also using the web based UI.

5.1 Breakpoint & Watchpoint

Breakpoint one of the most important and interactive debugging feature to the users. A spark program written by the user can be paused and resumed when user put a breakpoint between a set instructions. The idea here is to match the debugging experience that the users usually had in conventional debugging environment. Though both UI and programming API user can specify the breakpoint where he/she wants to put a breakpoint. Once placed, a user can run the program and it will start the computing the tasks at different nodes and once that instruction is reached where the breakpoint is placed the debugger will pause the execution right there. It is important to understand the granularity of this feature as the user write a Spark program at the RDD level which is a subtask that runs on each node on the allotted partition and the original input is partitioned into tasks which are assigned to worker nodes. A task can contain subtasks (only one-one dependency RDDs) which are actually the RDDs based instructions written by the user. Whenever there are one to many dependencies between RDD a stage boundary is created and the tasks before had to be completed before it can proceed with the next stage. Spark-BDD provide support to place breakpoints at all levels of physical and programming layer of a Spark program which includes stages, tasks and subtasks.

Pausing the execution itself at a certain point in the program does not provide valuable information about program. We needed some kind of a monitoring tool just like variable watching in normal debugging to watch the data flowing between stages and subtasks. Spark-BDD's Watchpoint catered that need by tapping the intermediate data flows at the breakpoint. User can see the data records monitored at that breakpoint and analyze the intermediate data to detect and eliminate the faults and errors. Considering the scale of these analytical programs the tapped data could in the order of giga bytes and keeping that data in memory is not a viable option as this data need to be showed to the user and since the user only interact with the one node (Driver), sending all this data over the network will cause tremendous overhead.

To avoid this overhead we implemented a filter mechanism for our monitor. Watchpoint filter mechanism is but on top of Scala and Java, so the user can actually write the filters like a normal spark program. For user to write this filter, he only need to write the implementation of a BDDPredicate class's filter function. The filter function takes in a Key-Value pair and returns a boolean value to indicate whether that record is to be sent to user or not. A user can provide this predicate implementation both in the spark program or in UI. Flexibility is also provided in the sense that user can dynamically write the filters and submit it into the on going Spark job. The debugger will then dynamically compile it and ship the compiled predicate to the workers. Since watch point is the feature that poses the highest overhead, we adapted a set of optimizations to minimize the network and disk traffic including batch sending, on demand watchpoint and volatile watchpoints. With watchpoint and breakpoint feature at disposal , a user can oversee the intermediate and run some simple and short analysis to detect the outliers , anomalies, faults or any kind of logical errors that can not be detected either through logs or by observing the original spark program.

5.2 Stragglers

Current version of spark provides a high level of information on how long did each work take to perform the task. But due to optimization most the transformation are done in a batch and current latency approach does not give you per transformation level latency so that you will exactly which transformation is causing trouble. To localize the abnormal behavior at the most finest level, we implemented straggler detection both at subtask and record level. User can enable the configurations for subtask or record level latency and get the execution/transformation times for each RDD and record respectively. Subtask level latency is important where the counts of records are very huge and most of computations are very basic, this will help in isolating the transformations and workers. Record level latency is crucial where time consuming transformation happen mostly on per record. This will help us identify the bad or outlying records that are delaying the job. User is usually given the information at

runtime that which of the transformations and records are straggling along with the physical layer information like information on workers. Worker based stragglers can provide some useful information,which consequently can help to perform runtime scheduling and shuffling optimizations.

5.3 CrashCulprit

5.4 Crash Recovery

5.5 Stragglers based reartitioning

6. EVALUATION

7. CONCLUSION

We have already seen several typeface changes in this sample. You can indicate italicized words or phrases in your text with the command `\textit`; emboldening with the command `\textbf` and typewriter-style (for instance, for computer code) with `\texttt`. But remember, you do not have to indicate typestyle changes when such changes are part of the *structural* elements of your article; for instance, the heading of this subsection will be in a sans serif[1] typeface, but that is handled by the document class file. Take care with the use of[2] the curly braces in typeface changes; they mark the beginning and end of the text that is to be in the different typeface.

You can use whatever symbols, accented characters, or non-English characters you need anywhere in your document; you can find a complete list of what is available in the *LaTeX User's Guide*[**?**].

7.1 Math Equations

You may want to display math equations in three distinct styles: inline, numbered or non-numbered display. Each of the three are discussed in the next sections.

7.1.1 Inline (In-text) Equations

A formula that appears in the running text is called an inline or in-text formula. It is produced by the **math** environment, which can be invoked with the usual `\begin. . .\end` construction or with the short form `$. . .$`. You can use any of the symbols and structures, from α to ω, available in LaTeX[**?**]; this section will simply show a few examples of in-text equations in context. Notice how this equation: $\lim_{n\to\infty} x = 0$, set here in in-line math style, looks slightly different when set in display style. (See next section).

7.1.2 Display Equations

A numbered display equation – one set off by vertical space from the text and centered horizontally – is produced by the **equation** environment. An unnumbered display equation is produced by the **displaymath** environment.

Again, in either environment, you can use any of the symbols and structures available in LaTeX; this section will just give a couple of examples of display equations in context. First, consider the equation, shown as an inline equation above:

$$\lim_{n\to\infty} x = 0 \tag{1}$$

[1]A third footnote, here. Let's make this a rather short one to see how it looks.
[2]A fourth, and last, footnote.

Table 1: Frequency of Special Characters

Non-English or Math	Frequency	Comments
Ø	1 in 1,000	For Swedish names
π	1 in 5	Common in math
$	4 in 5	Used in business
Ψ_1^2	1 in 40,000	Unexplained usage

Notice how it is formatted somewhat differently in the **displaymath** environment. Now, we'll enter an unnumbered equation:

$$\sum_{i=0}^{\infty} x + 1$$

and follow it with another numbered equation:

$$\sum_{i=0}^{\infty} x_i = \int_0^{\pi+2} f \tag{2}$$

just to demonstrate LaTeX's able handling of numbering.

7.2 Citations

Citations to articles [**?**, **?**, **?**, **?**], conference proceedings [**?**] or books [**?**, **?**] listed in the Bibliography section of your article will occur throughout the text of your article. You should use BibTeX to automatically produce this bibliography; you simply need to insert one of several citation commands with a key of the item cited in the proper location in the `.tex` file [**?**]. The key is a short reference you invent to uniquely identify each work; in this sample document, the key is the first author's surname and a word from the title. This identifying key is included with each item in the `.bib` file for your article.

The details of the construction of the `.bib` file are beyond the scope of this sample document, but more information can be found in the *Author's Guide*, and exhaustive details in the *LaTeX User's Guide*[**?**].

This article shows only the plainest form of the citation command, using `\cite`. This is what is stipulated in the SIGS style specifications. No other citation format is endorsed.

7.3 Tables

Because tables cannot be split across pages, the best placement for them is typically the top of the page nearest their initial cite. To ensure this proper "floating" placement of tables, use the environment **table** to enclose the table's contents and the table caption. The contents of the table itself must go in the **tabular** environment, to be aligned properly in rows and columns, with the desired horizontal and vertical rules. Again, detailed instructions on **tabular** material is found in the *LaTeX User's Guide*.

Immediately following this sentence is the point at which Table 1 is included in the input file; compare the placement of the table here with the table in the printed dvi output of this document.

To set a wider table, which takes up the whole width of the page's live area, use the environment **table*** to enclose the table's contents and the table caption. As with

Figure 1: A sample black and white graphic (.eps format).

Figure 2: A sample black and white graphic (.eps format) that has been resized with the `epsfig` command.

a single-column table, this wide table will "float" to a location deemed more desirable. Immediately following this sentence is the point at which Table 2 is included in the input file; again, it is instructive to compare the placement of the table here with the table in the printed dvi output of this document.

7.4 Figures

Like tables, figures cannot be split across pages; the best placement for them is typically the top or the bottom of the page nearest their initial cite. To ensure this proper "floating" placement of figures, use the environment **figure** to enclose the figure and its caption.

This sample document contains examples of **.eps** and **.ps** files to be displayable with LaTeX. More details on each of these is found in the *Author's Guide*.

As was the case with tables, you may want a figure that spans two columns. To do this, and still to ensure proper "floating" placement of tables, use the environment **figure*** to enclose the figure and its caption.

Note that either **.ps** or **.eps** formats are used; use the `\epsfig` or `\psfig` commands as appropriate for the different file types.

7.5 Theorem-like Constructs

Other common constructs that may occur in your article are the forms for logical constructs like theorems, axioms, corollaries and proofs. There are two forms, one produced by the command `\newtheorem` and the other by the command `\newdef`; perhaps the clearest and easiest way to distinguish them is to compare the two in the output of this sample document:

This uses the **theorem** environment, created by the `\newtheorem` command:

THEOREM 1. *Let f be continuous on $[a, b]$. If G is an antiderivative for f on $[a, b]$, then*

$$\int_a^b f(t)dt = G(b) - G(a).$$

Figure 3: A sample black and white graphic (.ps format) that has been resized with the `psfig` command.

The other uses the **definition** environment, created by the `\newdef` command:

Definition 1. If z is irrational, then by e^z we mean the unique number which has logarithm z:

$$\log e^z = z$$

Two lists of constructs that use one of these forms is given in the *Author's Guidelines*.

and don't forget to end the environment with figure*, not figure!

There is one other similar construct environment, which is already set up for you; i.e. you must *not* use a `\newdef` command to create it: the **proof** environment. Here is a example of its use:

PROOF. Suppose on the contrary there exists a real number L such that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = L.$$

Then

$$l = \lim_{x\to c} f(x) = \lim_{x\to c} \left[gx \cdot \frac{f(x)}{g(x)} \right] = \lim_{x\to c} g(x) \cdot \lim_{x\to c} \frac{f(x)}{g(x)} = 0 \cdot L = 0,$$

which contradicts our assumption that $l \neq 0$. □

Complete rules about using these environments and using the two different creation commands are in the *Author's Guide*; please consult it for more detailed instructions. If you need to use another construct, not listed therein, which you want to have the same formatting as the Theorem or the Definition[**?**] shown above, use the `\newtheorem` or the `\newdef` command, respectively, to create it.

A *Caveat* for the TeX Expert

Because you have just been given permission to use the `\newdef` command to create a new form, you might think you can use TeX's `\def` to create a new command: *Please refrain from doing this!* Remember that your LaTeX source code is primarily intended to create camera-ready copy, but may be converted to other forms – e.g. HTML. If you inadvertently omit some or all of the `\def`s recompilation will be, to say the least, problematic.

8. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or

Table 2: Some Typical Commands

Command	A Number	Comments
`\alignauthor`	100	Author alignment
`\numberofauthors`	200	Author enumeration
`\table`	300	For tables
`\table*`	400	For wider tables

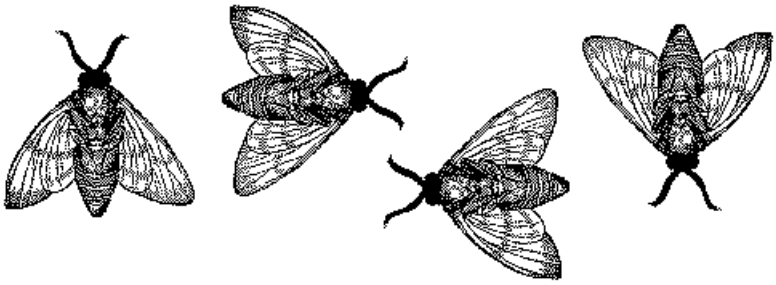

Figure 4: A sample black and white graphic (.eps format) that needs to span two columns of text.

Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LaTeX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

9. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

Inline (In-text) Equations

Display Equations

A.2.3 Citations

A.2.4 Tables

A.2.5 Figures

A.2.6 Theorem-like Constructs

A Caveat for the TeX Expert

A.3 Conclusions

A.4 Acknowledgments

A.5 Additional Authors

This section is inserted by LaTeX; you do not insert it. You just add the names and information in the `\additionalauthors` command at the start of the document.

A.6 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELP FOR THE HARDY

The acm_proc_article-sp document class file itself is chockfull of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LaTeX, you may find reading it useful but please remember not to change it.