iQCAR: A Demonstration of an Inter-Query Contention Analyzer for Cluster Computing Frameworks

Prajakta Kalmegh, Harrison Lundberg, Frederick Xu, Shivnath Babu, Sudeepa Roy
Department of Computer Science, Duke University
Durham, North Carolina
{pkalmegh,hgl2,hx26,shivnath,sudeepa}@cs.duke.edu

ABSTRACT

Unpredictability in query runtimes can arise in a shared cluster as a result of resource contentions caused by inter-query interactions. `iQCAR` - *i*nter **Q**uery **C**ontention **A**nalyze**R** is a system that formally models these interferences between concurrent queries and provides a framework to attribute blame for contentions. `iQCAR` leverages a multi-level directed acyclic graph called `iQC-Graph` to diagnose the aberrations in query schedules that lead to these resource contentions. The demonstration will enable users to perform a step-wise deep exploration of such resource contentions faced by a query at various stages of its execution. The interface will allow users to identify top-k victims and sources of contentions, diagnose high-contention nodes and resources in the cluster, and rank their impacts on the performance of a query. Users will also be able to navigate through a set of rules recommended by `iQCAR` to compare how application of each rule by the cluster scheduler resolves the contentions in subsequent executions.

KEYWORDS

Performance evaluation; contention analysis; blame attribution; resource bottleneck; cluster computing systems

ACM Reference Format:
Prajakta Kalmegh, Harrison Lundberg, Frederick Xu, Shivnath Babu, Sudeepa Roy. 2018. iQCAR: A Demonstration of an Inter-Query Contention Analyzer for Cluster Computing Frameworks. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183713.3193567

1 INTRODUCTION

Large scale data analytics frameworks like Hadoop [6] and Spark [11] process a mix of short-running interactive BI (Business Intelligence) queries along with long-running ETL or batch analytics queries. In such frameworks often recurring queries co-exist with adhoc unplanned queries. Moreover, analytical SQL queries with varying resource utilizations over time often share the cluster with machine learning, graph analytics, and data mining queries. In such shared clusters, resources are allocated to multiple tenants executing mixed workloads based on their priorities, SLAs (Service-Level Agreements), minimum share, etc. Typically, resource allocations are controlled by the cluster scheduler using sophisticated arbitration techniques like capped capacities, reservations or use of scheduling policies like `FAIR` [10] or First-In-First-Out (`FIFO`). Most of these techniques rely on partitioning of available slots[1] among the contending tenants. As a result, there are no guarantees on the usages of other resources like memory, disk IO, or network bandwidth for competing queries leading to inter-query resource interferences. This is a major concern in today's clusters as performance issues due to resource contentions are often wrongly diagnosed, or are left unresolved due to lack of appropriate tools. It is, thus, important to analyze the victims (that we call **target queries**) and sources of these contentions (that we call **source queries**) for identifying *why* and *where* a target query faces contentions from a source query. This can help a cluster administrator diagnose aberrations in resource allocations among tenants or devise alternative query placement strategies. For example, ranking the tenants based on their contention impact toward a target query can prove particularly useful to revisit the shares of resources for each tenant.

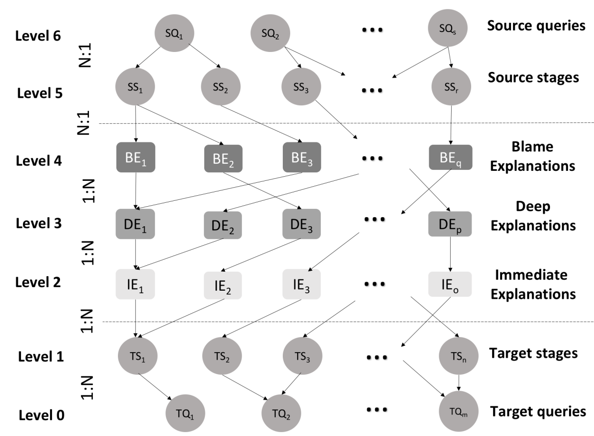


Figure 1: `iQC-Graph` with three levels of explanations.

In this demonstration, we will present `iQCAR` - **i**nter **Q**uery **C**ontention **A**nalyze**R**, a system to explore contentions faced by queries due to inter-query interactions on a cluster computing framework. `iQCAR` interface allows users to interact with a multi-level directed acyclic graph (DAG) to progressively unravel three levels of explanations; namely (i) **Immediate Explanations** (`IE`): identify disproportionate waiting times spent by a query blocked

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3193567

[1]We refer to a slot as the smallest unit of resource allocation. For example, its a CPU core in Spark and a combination of CPU and Memory in Hadoop.

for a particular resource, (ii) **Deep Explanations** (DE): inspect this waiting time for every resource used by the query on all hosts where it was executed, and finally (iii) **Blame Explanations** (BE): quantify the impact from concurrent queries toward the slowdown of a target query. Figure 1 presents different levels of explanations in iQC-Graph. Additionally, users will be able to filter and navigate through cluster-level summary visual aids on: (a) high contention resources and nodes, (b) high impact causing source queries, and (c) high impact receiving target queries. Finally, users will be able to browse through a list of alternate schedule rules recommended by iQCAR and compare the results of applying them in recurring execution of the workloads.

2 IQCAR SYSTEM

iQCAR automates the process of (i) collecting, parsing and persisting the query execution and cluster performance logs, (ii) construction and persistence of iQC-Graph, (iii) quantifying contention impact and blame attribution at various levels, and (iv) generation and application of rules for cluster scheduler to apply in subsequent execution of the workload. We present the multi-layered iQC-Graph in Section 2.1, and discuss how the architecture shown in Figure 2 facilitates each of these tasks in Section 2.2.

2.1 Multi-layered iQC-Graph

Level-0 of iQC-Graph consists of target queries to be analyzed and Level-1 contains the stages of each target query. Level-5 and Level-6 constitute the concurrently running source stages and source queries respectively. Levels 2, 3 and 4 of iQC-Graph provide explanations of different forms and granularity. They enables us to connect the two ends of iQC-Graph with appropriate assessment of contention impact among all intermediate nodes and edges.

2.1.1 Immediate Explanations (IE):. Level-2 vertices provide an explanation of the form '*how much time was spent by a stage waiting for a particular resource per unit of data processed*'. For every stage S_{tq} at Level-1 of target query Q_{tq}, we add an IE vertex at Level-2 for every resource (scheduling queue, CPU, Memory, Network, IO) used by stage S_{tq}, and store the value of its wait-time for this resource per unit data processed as its Vertex Contribution (VC).

2.1.2 Level-3: Deep Explanations (DE):. Level-3 captures the *hosts* responsible toward the corresponding disproportionality in the wait time components for every resource. That is, DEs keep track of the wait-time distributions per unit of data processed by stage S_{tq} for a specific resource r on each host h of execution. That is, for each IE node in Level 2, we add h DE nodes in Level 3 corresponding to all hosts on which the tasks of S_{tq} executed.

2.1.3 Level-4: Blame Explanations (BE): . Blame Explanations is a novel contribution of iQCAR. For each vertex v in Level-3 (DE) corresponding to S_{tq}, host h, and type of resource request r, if tasks of S_{tq} were concurrent with tasks of P stages of other queries on host h, we add P nodes u in Level 4 and connect them to v. To compute the blame to be assigned to a concurrent stage, we first compute the blame from a source task (task of concurrent stage) to a target task executing concurrently on host h while using resource r and use it to compute the VC of each BE node.

2.2 Architecture

iQCAR provides a cluster interface to analyze existing Spark workload execution logs, submit new Spark applications, or simulate an existing workload using TPCDS [5] benchmark queries.

wlSEL and wlSUB: The workload selection module (wlSEL) allows users to select a pre-executed workload for analysis. To let users simulate a workload on Spark, the workload submission interface (wlSUB) allows them to select a list of benchmark TPCDS queries, the order of these queries and finally their arrival schedule (fixed-delay or poisson or user-input start times). The users can also specify the interval (in seconds) for collecting the task execution metrics. By default, we collect metrics after the completion of each task, stage, job, or query in Spark.

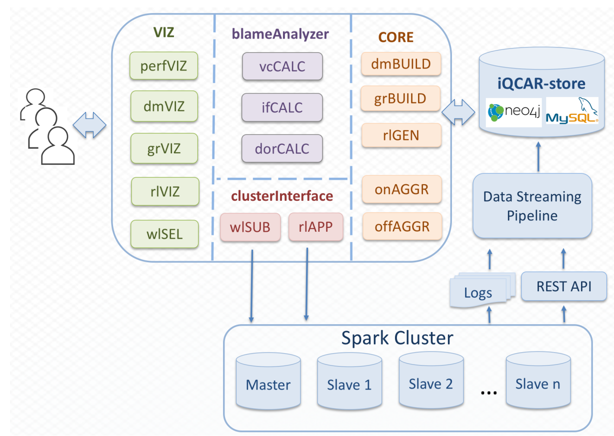


Figure 2: iQCAR Architecture

iQCAR CORE: The offAGGR module collects and aggregates cluster logs for queries executed on Spark offline, and parses them. For an online execution analysis, the onAGGR module collects the execution metrics using the REST API interface of Spark [2] and streams them through our streaming module to a MySQL database. The data model builder module (dmBUILD) uses this input to build the data model for iQCAR. An admin can configure whether to persist this data model in a CSV format or MySQL store. By default, we store in a CSV format for later easy integration with our iQCARViz API. iQCAR also provides an easy export from our MySQL store to the iQCARViz data frames and series objects. Users use hints from the iQCARViz interface (described shortly) to select a list of target queries for deep exploration. The graph builder module (grBUILD) uses our parallel graph construction algorithm (see [7]) to build iQC-Graph using the Neo4j graph API [2]. By default, we persist a Neo4j graph instance for every workload, and reload the graph when user requests for a deep exploration of the selected target queries through the iQCARViz interface.

blameAnalyzer: A graph-based model enables us to consolidate the contention and blame values for a systematic deep exploration. To enable a comparison of contention impacts at various levels of iQC-Graph, the blameAnalyzer consists of three modules that

[2]We extended the Spark metrics accumulator API to publish our custom wait-time metrics for all resources (scheduling delay, CPU, Network, Memory, and IO blocked times) to the REST interface.

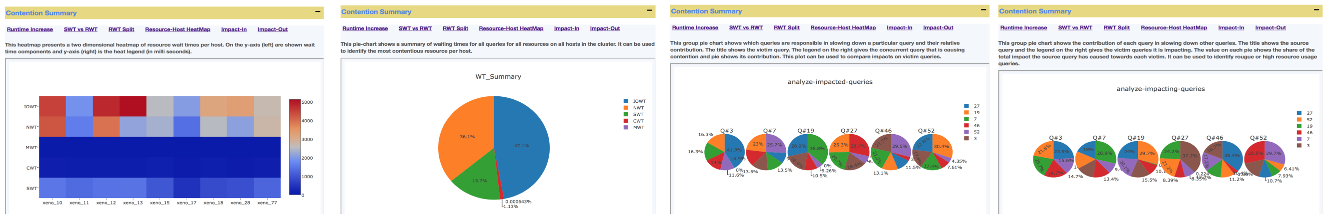


Figure 3: `iQCAR` visual aids for answering cluster-level contention analysis questions.

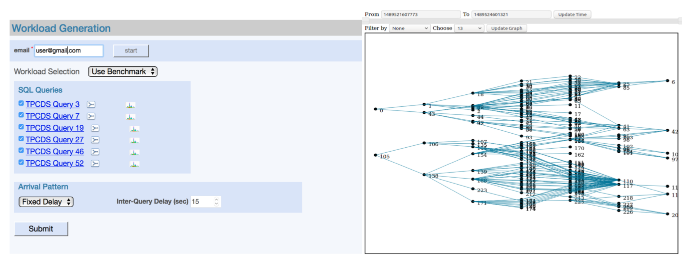
calculate the following contention measures: The Vertex Contribution (`VC`) values are computed using the `VC`-*calc* that measures the standalone impact of any vertex toward the contention faced by a target query. The `VC` values of different vertices depend on the level to which the vertex belongs to, and are computed during the graph construction process as described in Section 2.1. We then perform a top-down pass on this graph to update the edge weights using the `IF`-*calc*, and next do a bottom-up pass to update the responsibility measure of each vertex using the `DOR`-*calc*. The *Impact Factor* (`IF`) of an edge gives the impact a source vertex of the edge has toward the end vertex. *Degree of Responsibility* (`DOR`) of a vertex is defined as the cumulative impact this vertex has toward a target query.

`rlGEN` **and** `rlAPP`**:** The rule generator (`rlGEN`) module uses the consolidated contention measures from **blameAnalyzer** and outputs two types of rules: (i) Alternate Query Placement: `rlGEN` outputs the top-k aggressive queries (high-impact toward all selected target queries) and generates k rules that recommend placing each of these queries in a new pool with new recommended shares for each of them. (ii) QueryPriority Readjustment: `rlGEN` produces k priority rectification rules for each of the top-k affected target queries that suffered the highest impact, and top-k impacting source queries. The `rlAPP` module is an extension to the Spark standalone scheduler that parses the rules and applies any active and applicable rules during a recurring execution of the workload.

`iQCAR` **VIZ :** The `iQCAR` visualization module is a web based front-end that enables users to (i) select a workload for analysis using our `wlSEL` module and visualize its key characteristics using the `wlVIZ` interface, (ii) explore summary of query execution and cluster performance using the `perfVIZ` interface that provides tips on selecting a single target query or a set of target queries for further exploration, (iii) delve into the task-level execution and wait-time distribution details for each query using the `dmVIZ` interface, (iv) perform a systematic deep exploration of the Neo4j `iQC-Graph` that lets users unfold explanations at various levels and analyze the relative impacts from concurrent queries using the `grVIZ` visualization aids, and (v) finally, use the `rlVIZ` module that lets users compare the results of applying a selected set of rules on the recurring execution of the workload. The `iQCARViz` interface also allows users to compare the impact of different monitoring intervals of data collection on our contention analysis metrics. Each of the visualization modules use the `iQCARViz` dataframes API to render plots dynamically based on user-input online using Plotly [3] tool.

3 DEMONSTRATION

The purpose of this demonstration is to (i) showcase the users the tedious process of contention analysis in the absence of `iQCAR`, (ii) enable users with a hands-on experience of using `iQCAR` for insightful analysis, and (iii) present users an opportunity to compare and contrast the results of applying the `iQCAR` rules on a benchmark workload's performance. To achieve these goals, we will divide the demo in three segments, namely (a) manual analysis of pre-executed TPCDS benchmark execution, (b) deep exploration of contentions using `iQCAR`, and (c) analyze the output of `iQCAR`.



(a) (b)

Figure 4: (a) Screen to select a workload for contention analysis. (b) Screen to perform a time-series analysis of `iQC-Graph`.

Setup: Users will be given two options to analyze a workload: (i) select a pre-executed microbenchmark workload, or (ii) submit a workload by selecting a set of TPCDS queries along with their arrival pattern. A sample screen for this workload selection is shown in Figure 4a. The workloads will be executed on a 10-node cluster setup with Apache Spark 2.2 [11] and Hadoop 2.7 [6].

3.1 Segment 1: Manual Analysis

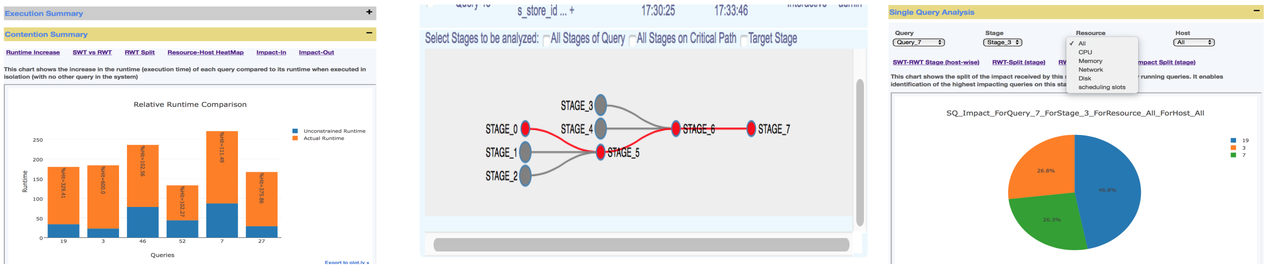
Users will be asked to answer one randomly selected multi-choice question on the contention faced by a query using the existing monitoring tools like Spark UI [4] and Ganglia [1]. This activity will demonstrate the tedious process of performing a manual contention analysis even on a small-size cluster.

3.2 Segment 2: Explore `iQCAR`

The next step in our demo is to explore the interface of `iQCAR` that will enable users answer questions like below divided broadly into the following three categories based on the level of contention details they provide:

Summary Questions: Users can choose to browse through a series of cluster summary visual aids for our sample workloads illustrated in Figure 3.

- **Q1:** Which hosts in the cluster had the highest CPU contention?
- **Q2:** On which resource were all queries bottlenecked the most?
- **Q3:** Which queries are the victim of highest contention?



(a) (b) (c)

Figure 5: (a) Screen to aid users identify queries for further exploration. It shows the runtime hit of all queries compared to their unconstrained (no contention) execution time. (b) Screen to aid users identify stages of a selected query for further exploration. (c) `iQCAR` visualization to select combinations of a host, resource and source queries to analyze impact on a single selected target query.

- **Q4:** Which queries are the cause of highest contention?

Target Query Performance Analysis Questions: Figure 5a shows how `iQCAR` provides a visual aid for users to select a target query for further deep exploration from a set of completed queries in a workload. Next, Figure 5b shows how users can choose whether they want to analyze contentions on (i) a single stage, (ii) all stages on the critical path (execution time dominating path), or (iii) all stages of a query using `iQCAR`. For performing a single-query analysis, users can drill-down to various levels of details by filtering on hosts, resources or specific stages of target and source queries as shown in Figure 5c to explore the impact on a particular target query Q_t. This interface allows users to answer questions like:

- **Q5:** On which user-selected combination of host h and resource r, has a target query Q_t (or its selected stages) spent maximum time waiting?
- **Q6:** Which queries are responsible for causing highest contention for a target query Q_t (or its selected stages) on a user-selected combination of host h and resource r?

Source Query Performance Analysis Questions: Finally, users can also perform a top-down analysis on `iQC-Graph` to draw insights on how a query impacts or causes contentions to others using various filters on its stages, hosts, and resource types. Due to space constraints, the screenshots for these visualizations are not shown.

- **Q7:** Which queries were affected most by the contention caused by a source query Q_s (or its selected stages) on a user-selected combination of host h and resource r?

The above visual aids help users get answers to questions Q1 to Q7 rapidly. For users who want to diagnose each contention in detail, the `iQC-Graph` visualization provides a step-wise exploration opportunity. For instance, users can click on each vertex and edge to view the values of our contention analysis metrics (`VC`, `IF`, and `DOR`). Other interface features will let the users (i) highlight the path from a single source query to a target query with highest path weight (useful for providing explanations for highest impact between any two queries), (ii) display all paths with path weights crossing a certain threshold of user-input impact value (useful to discover contention conditions beyond an acceptable threshold), and (iii) load source queries that impact the selected target queries only within an user-input time frame. Figure 4b shows the `iQC-Graph` for our example workload for the last scenario where user inputs the start and end times to input a time frame for impact analysis.

3.3 Segment 3: Analyze `iQCAR` results

The final segment in the demo will enable users to examine a set of rules output by the `rlGEN` module of `iQCAR`. Users will be able to compare the performance of the workloads (new runtime of each query, wait-times on all resources and/or hosts) after application of the top-3 rules of each type. Users can also use the recommended priority to choose and apply the rules for a more real-time experience.

Related Work: The field of explanations has been studied in many contexts like analyzing job performance [8]. In [9], the authors present a general framework to analyze data-analytical workloads using *blocked-time* metric. We use this pedestal to present `iQCAR` as a first systematic tool toward exploration of different levels of explanations for resource contentions on cluster frameworks. A detailed discussion on related work is presented in [7].

Acknowledgment. *This work was supported in part by NSF awards IIS-1423124, IIS-1552538, IIS-1703431, and NIH Award 1R01EB025021-01.*

REFERENCES

[1] 2018. Ganglia Monitoring System. http://ganglia.info. (2018).
[2] 2018. Neo4j: A graph database. https://neo4j.com. (2018).
[3] 2018. Plotly: Modern Visualization for the Data Era. https://plot.ly. (2018).
[4] 2018. Spark Monitoring and Instrumentation. http://spark.apache.org/docs/latest/monitoring.html. (2018).
[5] 2018. TPC Benchmark™DS . http://www.tpc.org/tpcds/. (2018).
[6] Doug Cutting. 2018. Apache Hadoop. http://hadoop.apache.org. (2018).
[7] Prajakta Kalmegh, Shivnath Babu, and Sudeepa Roy. 2017. Analyzing Query Performance and Attributing Blame for Contentions in a Cluster Computing Framework. *CoRR* abs/1708.08435 (2017). arXiv:1708.08435 http://arxiv.org/abs/1708.08435
[8] Nodira Khoussainova, Magdalena Balazinska, and Dan Suciu. 2012. Perfxplain: debugging mapreduce job performance. *PVLDB* 5, 7 (2012), 598–609.
[9] Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. 2015. Making Sense of Performance in Data Analytics Frameworks. In *NSDI*. 293–307. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/ousterhout
[10] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. 2010. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*. ACM, 265–278.
[11] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets *(HotCloud)*. 10–10.