# Wave Computing in the Cloud

Bingsheng He[†] Mao Yang[†] Zhenyu Guo[†] Rishan Chen[†‡]
Wei Lin[†] Bing Su[†] Hongyi Wang[†] Lidong Zhou[†]
[†]*Microsoft Research Asia* [‡]*Beijing University*

## ABSTRACT

We introduce the new *Wave* model for exposing the temporal relationship among the queries in data-intensive distributed computing. The model defines the notion of *query series* to capture the recurrent nature of batched computation on periodically updated input streams. This seemingly simple concept captures a significant portion of the queries we observed in a production system. The recurring nature of the computation on the same input stream opens up surprisingly significant opportunities for achieving better performance and higher resource utilization.

## 1 INTRODUCTION

Recent work on data-intensive distributed computing (e.g., MapReduce [4], Dryad [7], and Hadoop [6]) has enabled large-scale data analysis as *a query* to execute in parallel on a large cluster of machines, despite failures during the computation. While the emergence of high-level languages, such as Sawzall [11], Pig [9], SCOPE [3], and DryadLINQ [14], has further reduced programming complexity, the research remains largely centered on individual queries. In reality, we are facing the challenging system problem of executing a large number of potentially complicated queries on a large amount of data every day on a large-scale cluster. Questions naturally arise: is the system doing a good job of utilizing the resources fully? Is the system executing the queries in a globally optimal way? We have not yet been able to answer such basic system questions satisfactorily or even to define the system goals precisely.

Our experience with a production computing cluster shows that we are far from reaching the ideal. For example, in the cluster we investigate, we have seen significant redundancy in computation across queries; that is, the same computation is performed multiple times on the same data for different queries, resulting in wasted I/O and computation. Load imbalance is also evident over time with periods of system overload and periods of resource under-utilization. Those can be attributed to inadequate data and resource management in the system.

Performance and resource optimization through the management of data and resources has been studied extensively in databases systems and (distributed) operating systems for decades. It is natural for us to look for solutions and inspirations from those fields, as proposed by Olston et al. [1, 8]. For example, the notion of *views* in databases and the *view matching* [15] techniques are particularly effective in identifying common computations or sub-computations across queries and in allowing the results to be reused.

While leveraging the proven concepts in the fields such as databases is clearly a step in the right direction, applying those concepts in the current computing environment itself is particularly challenging due to the inherent complexity and unpredictability in the system. For example, query optimization in databases hinges on a cost model. For a query in our environment, the system often has little knowledge about the data being processed; a query could use custom functions with unknown performance characteristics; a query is often complicated and contains sub-queries, resulting in computation consisting of multiple distributed *steps*. All those make a reliable cost model nearly impossible.

With challenges also come opportunities. We observe that log data mining has been the original motivation for such data-intensive distributed computing systems and it remains a dominant workload in such systems. We therefore introduce a new *Wave* model that captures the key properties of log mining. In the Wave computing, we model the data not as a static file, but as a *stream* that is periodically updated. The stream is append-only and partitioned on multiple machines. A segment is the data from a single bulk update, e.g., the daily generated log. We further define the notion of *query series* to refer to recurrent computations on a stream, with each performed on one or more stream segments. Query series captures a sequence of the same computation on different sets of segments of the same stream and explicitly exposes the correlations among the queries in the query series in terms of both data and computation.

This seemingly simple notion of query series brings predictability into the system, and opens up new research opportunities by making previously unsolvable problems tractable. For example, with query series, the system knows the queries that need to be executed as data streams are updated. Query series makes the occurrence of these queries predictable. This offers flexibility in the scheduling decisions: Queries in different query series might share the same I/O to scan the input data and might even share common computation. Those queries could be scheduled to run together as a single combined query by removing redundancies. Furthermore, query series makes the construction of a reliable cost model a

possibility by leveraging the knowledge of data and computation from the executions of the previous queries in the same query series. Data distribution within a stream tends not to change when the stream grows over time. Knowledge about the data collected from the executions of the early queries in a query series could provide excellent hints for the distribution of the data in the stream and allow later queries to be optimized accordingly. Previous executions in the same query series could help predict the cost of custom functions used in the recurring computation, as well as the cost of computation in each individual step.

**Organization**. The remainder of this paper is organized as follows. To exemplify the problems and opportunities in the current systems and to justify the Wave computing model, Section 2 presents the preliminary results on a query trace in a production cluster. We then outline the research directions for enabling Wave computing in Section 3. Finally, we present the concluding remarks in Section 4.

## 2 WAVES IN THE CLOUD: PRELIMINARY STUDIES

To confirm the problems and the opportunities, as well as the dominance of the Wave-like patterns in the current system, we studied a query trace obtained from a production cluster. The queries are written in SCOPE [3], a declarative scripting language designed for massive data analysis. The query trace stores the basic information related to the execution of each query in the system, including the query itself, the submission time, the query plan, the performance statistics, such as I/O of each step in the query plan, and the completion status. The trace contains nearly 20 thousands of successfully executed queries, taking a total of 29 millions of machine hours. These queries are on around 140 data streams stored in a reliable append-only distributed file system.

**Redundancy.** We first studied the redundant operations across queries in the query trace. Specifically, we identified two kinds of redundant operations: input data scans and common sub-query computation.

Redundant I/O for scanning the input files are common in the cluster. For all the query executions in the trace, the I/O of scanning the input files contributes to about 66% of the total I/O. The total size of the input files is about 33% of the total I/O. Thus, the redundant I/O on scanning the input files contributes to around 33% of the total I/O, causing significant waste in the disk bandwidth. The most frequently accessed input stream is accessed more than 200 times on average per day.

Figure 1 illustrates the submission day and the input data window of three sample query series on the same stream in August. Query series 2 and 3 have recurring computation on a per-day data window, and the inputs of queries in query series 1 overlap. Comparing the submission days of different queries with the same input data window, we find that these queries are sometimes submitted on the same day, and others on different days, which results in redundant I/O scans. Examples are highlighted in Circle **A**.

Redundant computation on common sub-queries is also significant. A query execution is divided into multiple *steps*. We sort all the queries in the trace into a sequence based on the query submission time. A step $s$ is defined to have a *match* if there exists a step $s'$ of a previous query in the sequence, where $s$ and $s'$ have the same input and a common computation. Each step with a match is redundant computation because the same computation has been performed on the same data previously. We consider all the successful queries and find that 30% of the steps have a match. In Figure 1, since queries in query series 1 have common computation on the overlapping input windows, there is often redundant computation among them, as highlighted in Circle **B**.

**Load Imbalance.** Our next step is to study the temporal distribution of the workload in the production cluster. Figure 2 shows the normalized total machine time for all query executions per day in August. The total machine time fluctuates with a certain pattern: the total machine time in weekdays is on average 50% higher than that in weekends. The temporal load imbalance results in resource utilization problems including contention and under-utilization. In Figure 1, comparing the submission day and the input data window of each query, we find that their submission day is not aligned with the data window. One example is highlighted using Circle **C**. Some daily queries for the data in those three days are delayed due to a weekend. This results in the load imbalance that we have seen in Figure 2.

**Success Rate vs. Window Size.** We further observed that the success rate of query executions is affected by the input window size. The window size[1] of a query is defined to be the size of the time-window of the query's input on the stream. Table 1 shows the success rate of query executions categorized in their window sizes. As the window size increases, the input data size becomes large, and the query execution is more likely to fail, often due to resource contentions or exhaustion. This is also consistent with the findings in a previous study [11]. Due to the lower success rate, the queries with large input windows tend to reduce the effective resource utilization significantly.

**Waves in the Cloud.** We studied the recurring com-

[1]Theoretically, the input of a query such as the one consisting of a join operation can have multiple streams with different time windows. In practice, the input streams of most queries align to the same time window.

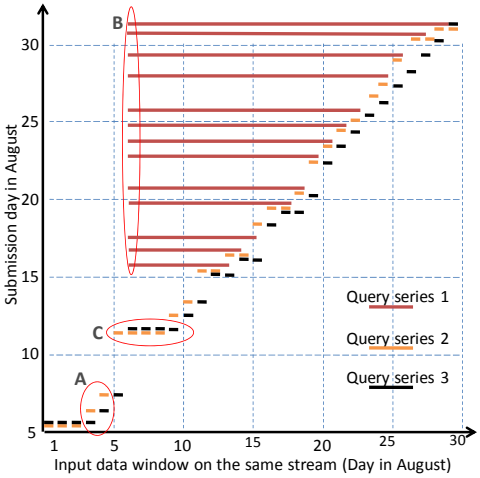


**Figure 1**: Sample query series on the same stream

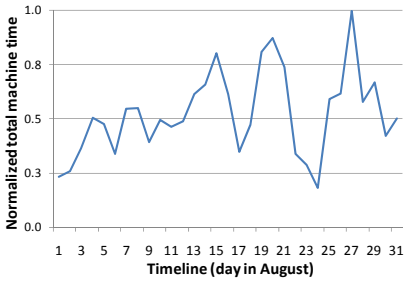


**Figure 2**: Daily total machine time

putation in the query trace to look for the Wave pattern. We looked at around 20 thousand queries that were successfully executed. These queries can be categorized into around 1100 query series. Over 95% of the query series have at least two queries. Over 74% of the query series are performed on the per-day input data window, and over 14% on the per-month input data window. There are in total 143 streams accessed around 40 thousand times. The top ten accessed streams have around 75% of the total number of accesses. The update is appended to the stream daily or when the update reaches a predefined threshold in terms of size. Thus, in the query trace, recurring computation is common and a small number of streams are frequently accessed. The Wave model therefore matches well with the computation needs of the production cluster.

**Table 1**: Success rate of queries with different window sizes

| Window size | one day | one week | one month | one quarter | six months | one year |
|---|---|---|---|---|---|---|
| Success rate | 90% | 78% | 75% | 58% | 52% | 6% |

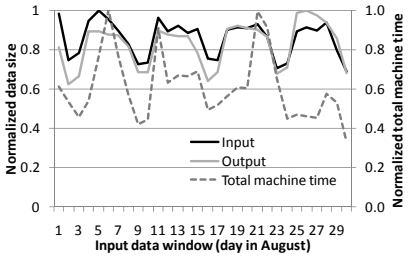


**Figure 3**: Normalized input and output data sizes for query series 2

**Predictability.** We further validated the similarities among the executions of the queries in the same query series. Figure 3 shows the normalized input and output data sizes, and the normalized total machine time of the queries in query series 2 of Figure 1. We normalize the value according to the maximum value of its kind during the period. For each query, the output data is obtained through applying the same set of filters on the input data. The output data size is clearly correlated with the input data size, thereby providing excellent hints on the filtering ratio of the filters. Furthermore, the total machine time is also well correlated to the input data size. These results show promises in predicting the behavior of the executions of later queries in a query series from the previous ones.

## 3 Waves of Opportunities

The Wave model opens up potential opportunities, but research challenges remain on how to turn those opportunities into better data-intensive distributed systems. We look at how the system can enable predictions, leverage the predictions in a variety of optimizations, and discuss their implications on the current systems.

### 3.1 Enabling predictions

When executing a query in a query series, the defining characteristics of the execution are captured and stored for prediction purposes. The characteristics fall into the following categories: (a) the input and output data characteristics including sizes and distribution, (b) the computation complexity of the operation such as the custom function, and (c) the cluster execution environment such as the network topology and computation resource configuration. The system collects statistics on the first two kinds of factors for each execution, and stores the statistics of the cloud environment as constants. All these statistics are stored in a *catalog*. Because the predictions are used to decide between the different options of executing the queries, transient behaviors such as failures during the execution are not captured or modeled for the prediction.

Data distribution of the stream tends to be important in a variety of optimization decisions, especially considering that the stream will be processed many times. It is sometimes advantageous to piggyback the collection of the extra statistics of the stream into the execution of an earlier query in a query series.

Collecting the data is the easy part; the challenge is to formulate a model for the execution behavior of the operation, which involves modeling the network, the distributed storage system, and other components in the cluster, as well as the behavior of the individual steps in the query. The benefit of the Wave model is that recurring execution can provide the knowledge of the query execution behavior and data properties on the same computation and the same data stream.

The key is to identify the characteristics that remain largely unchanged across the queries in the same query series and reuse those in the prediction for the later queries in the same query series. Such stable characteristics help model the cost of query processing based on the previous executions of the query series. Previous work on cost modeling in databases, both those using an analytical model [10] and those using a machine learning approach [5], could provide useful insights, even though the Wave model has simplified the problem and made possible more accurate predictions.

### 3.2 Wave optimizations

The Wave model enables a set of optimizations and provides predictions that help the system make the appropriate choices.

**Shared Scan and Computation.** Queries from a set of different query series might be reading from the same input stream segments at the same schedule or even contain common computation on the same data segments. Because the occurrences of those queries are known in advance in the Wave model, those queries can be folded into a single large query with a single scan on the input and without redundant computations in the sub-queries. Identifying opportunities for shared scans is often straightforward as the input stream segments for each query are precisely specified and can be matched easily. (Note that, with the Wave model, the system no longer needs to build a stochastic model as proposed by Agrawal et al. [1].) Research in databases, such as those on incremental computation [12] and on view matching [15], can help discover common sub-queries.

**Query Decomposition.** A query might be decomposed into a series of smaller queries each on a subset of input stream segments, followed by a final step of aggregating the results of the smaller queries to obtain the final result. Not all queries can be decomposed easily without significantly increasing the overall complexity of the execution, but many queries, such as computing the histogram, can. Often, the decomposability can be decided based on the properties of the operators and the custom functions used in a query. For those that can be decomposed, query decomposition could be beneficial in the following three ways.

- Query decomposition might help uncover more opportunities for shared scan and computation. For example, if the decomposition makes all queries on the same stream process the data on aligned daily windows, there would clearly be more opportunities for sharing among the queries.
- Query decomposition can alleviate the load imbalance. For example, for a query operating on a data window of a month, it can be decomposed to a series of daily queries, followed by a final aggregation query for the final result. The load of that query is therefore no longer concentrated at the end of the month, but spread over the duration of the month.
- Query decomposition can potentially help improve the success rate of the queries by reducing the size of each individual query, as indicated in Table 1. Further investigation is needed to understand the detailed root causes of the query failures and to assess whether decomposition truly helps.

**Query Planning.** A query can be executed in a variety of ways; query planning aims at finding the optimal way to execute a query. This is a traditional database problem. In our distributed execution environment, another dimension of choices are available; for example, in terms of the number of servers to use in each step of the computation and the locations of those servers. The consideration of the opportunities for shared scan/computation and for decomposition further complicates the process. That said, the predictability offered by the Wave model can potentially allow the system to zoom in on a small set of choices rather quickly. And the system can converge to the optimal query plan as more and more queries get executed in the same query series.

**Query Scheduling.** With multiple queries to be executed, the system must schedule the queries for best performance in terms of resource utilization and query completion time. Overall, the predictions from the Wave model can potentially lead to better schedules. Besides the traditional considerations such as priorities and query dependencies, the Wave model offers new options and challenges.

In order to exploit shared scan and computation, the system tends to bundle a number of queries from different query series together for better resource utilization. This is not without drawbacks. The resulting *jumbo-query* after bundling tends to exacerbate the load imbalance in the system. The scheduling mechanism should

preserve the semantics of individual queries in scheduling the jumbo-query. For example, in order to reduce the response time of individual queries, the system should provide flexibility in scheduling a specific query. Moreover, the system should provides a query-level fault tolerance mechanism. When a query fails, fault tolerance mechanisms are performed on the failed query only, and other queries can continue their executions.

### 3.3 Waves into the cloud

A typical workload will likely consist of those conforming to the Wave model and those *ad hoc* queries that do not. The support for the Wave model can be built on top of the existing systems, thereby leveraging the the capabilities such as failure handling, single query compilation/optimizations, and job scheduling. To support the Wave model and enable the optimizations, the system should (i) extend the language for describing query series, (ii) incorporate a cost model with useful statistics captured in a catalog, and (iii) enable a variety of query rewriting for cross-query optimizations. Examples of query rewriting include query decomposition and that of combining multiple queries into a jumbo-query with reduced redundancies. Ad hoc queries can also benefit from these mechanisms; for example, by leveraging the statistics and the cost model for query optimization.

From our preliminary experiences of incorporating the Wave model into an existing system, such as SCOPE and DryadLINQ, we have found that we can benefit greatly from a high-level declarative language, especially through query rewriting; compared to the well-defined operators in the relational algebra, arbitrary custom functions supported in those systems tend to reduce the optimization opportunities. Furthermore, although the concept of cost model and query rewriting has been proposed and extensively studied in database systems, the unique characteristics in the cloud demand significantly different designs and implementations.

## 4 Conclusion

Wave computing introduces a simple new model that sits between the traditional batch processing [13] and the stream processing models [2]. While the data is considered as streams that are constantly updated, the updates are persisted and available. As a result, the periodic processing on the stream is of the batch nature, without the resource and time constraints normally found in stream processing systems. And unlike batch processing that looks at individual queries, the Wave model defines a series of correlated queries.

Wave computing is practically interesting because it matches a large portion of the workload on the current systems and is practically feasible because it can be largely enabled on top of the existing systems. Despite its simplicity, by capturing the correlations among queries in terms of the data and the computation, the Wave model can potentially further unlock the full power of data-intensive distributed computing. The Wave model fosters a set of interesting new research directions; the research is likely to yield exciting results that go beyond the Wave model itself, further advancing the state of the art.


### Acknowledgements

We would like to thank the Cosmos team for the access to the query trace. We are also grateful to Yuan Yu, Jingren Zhou, Li Zhuang, the System Research Group of Microsoft Research Asia, as well as the anonymous reviewers, for their insightful comments.


## References


[1] P. Agrawal, D. Kifer, and C. Olston. Scheduling shared scans of large data files. *Proc. VLDB Endow.*, 1(1):958–969, 2008.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *ACM PODS*, 2002.

[3] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: easy and efficient parallel processing of massive data sets. *Proc. VLDB Endow.*, 1(2), 2008.

[4] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[5] A. Ganapathi, H. Kuno, U. Dayal, J. Wiener, A. Fox, M. Jordan, and D. Patterson. Predicting multiple performance metrics for queries: Better decisions enabled by machine learning. In *IEEE ICDE*, 2009.

[6] Hadoop. *http://hadoop.apache.org/*.

[7] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst. Rev.*, 41(3):59–72, 2007.

[8] C. Olston, B. Reed, A. Silberstein, and U. Srivastava. Automatic optimization of parallel dataflow programs. In *USENIX ATC*, 2008.

[9] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: a not-so-foreign language for data processing. In *ACM SIGMOD*, 2008.

[10] M. T. Ozsu and P. Valduriez. *Principles of distributed database systems*. Prentice-Hall, Inc., 1991.

[11] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data: Parallel analysis with Sawzall. *Sci. Program.*, 13(4), 2005.

[12] G. Ramalingam and T. Reps. A categorized bibliography on incremental computation. In *POPL*, 1993.

[13] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. *SIGMOD Rec.*, 29(2), 2000.

[14] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. K. Gunda, and J. Currey. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *OSDI*, 2008.

[15] J. Zhou, P.-A. Larson, J.-C. Freytag, and W. Lehner. Efficient exploitation of similar subexpressions for query processing. In *ACM SIGMOD*, 2007.