Design and Implementation of a Real-Time Interactive Analytics System for Large Spatio-Temporal Data

Shiming Zhang[*] Yin Yang[#] Wei Fan[*] Marianne Winslett[#]

[*]Huawei Noah's Ark Lab

{simon.zhangsm, david.fanwei}@huawei.com

[#]Advanced Digital Sciences Center, University of Illinois at Urbana-Champaign

{yini, winslett}@illinois.edu

ABSTRACT

In real-time interactive data analytics, the user expects to receive the results of each query within a short time period such as seconds. This is especially challenging when the data is big (e.g., on the scale of petabytes), and the analytics system runs on top of cloud infrastructure (e.g., thousands of interconnected commodity servers). We have been building such a system, called OceanRT, for managing large spatio-temporal data such as call logs and mobile web browsing records collected by a telecommunication company. Although there already exist systems for querying big data in real time, OceanRT's performance stands out due to several novel designs and components that address key efficiency and scalability issues that were largely overlooked in existing systems. First, OceanRT makes extensive use of software RDMA one-sided operations, which reduce networking costs without requiring specialized hardware. Second, OceanRT exploits the parallel computing capabilities of each node in the cloud through a novel architecture consisting of Access-Query Engines (AQEs) connected with minimal overhead. Third, OceanRT contains a novel storage scheme that optimizes for queries with joins and multi-dimensional selections, which are common for large spatio-temporal data. Experiments using the TPC-DS benchmark show that OceanRT is usually more than an order of magnitude faster than the current state-of-the-art systems.

1. INTRODUCTION

Recently, considerable research efforts have been devoted to the infrastructure for *real-time interactive analytics* (or simply *real-time analytics*) over massive amounts of data. For example, in a typical interactive session, a user submits a query, waits online for its results, and possibly issue additional queries based on these results. The queries in such applications are usually exploratory in nature, for which the user expects to receive results quickly, e.g., within seconds. Meanwhile, the user may use business intelligence (BI) tools to generate queries automatically. As pointed out in [5], many BI tools expect the underlying data management system to return results within a short period of time; otherwise, they may timeout and return an error. For these applications, it is critical to achieve a low response time, meaning that offline, batch-oriented analytics tools are unfit for this purpose. Our goal is to perform real-time analytics over big spatio-temporal data, such as call logs and mobile web browsing records from a large mobile network operator [30]. Naturally, queries on such data often contain temporal and/or spatial predicates. Towards this goal, we have been building OceanRT (first introduced in [30]), a cloud-based system that aims for high scalability and resource efficiency, while ensuring low query response time.

We attribute OceanRT's high performance in the experiments in this paper to three novel designs, in its networking, computing architecture, and storage scheme, respectively. First, profiling of existing systems reveals that network transmissions incurs a significant cost in cloud-based systems in general, and real-time analytics systems in particular, which typically require the cooperation of a large number of computing nodes to answer a query, in order to satisfy the real-time response time requirements. OceanRT's solution is to employ remote direct memory access (RDMA) [23] extensively instead of traditional socket-based transmission methods. In particular, as we describe in Section 3.2, OceanRT employs a software implementation of RDMA, which accelerates network transmissions without requiring costly, specialized hardware.

Second, the design of OceanRT captures the parallel computing capabilities of modern servers which typically have multiple CPU cores and hard drives, as well as a large amount of RAM and Flash storage. While it is possible to subdivide each server into virtual machines, doing so incurs high overhead. OceanRT's computing architecture, which includes multiple interconnected Access-Query Engines (AQEs) in each node, ensures high parallelism with minimal overhead.

Third, data storage in OceanRT is optimized for speed without losing the fault tolerance and load balancing properties of traditional cloud-based storage systems. Meanwhile, its file layout and indexing are optimized for queries with joins and/or multi-dimensional range selections, which are common for spatio-temporal data.

An overview of a preliminary version of OceanRT was presented in a demo [30], which showed that OceanRT was faster than the state-of-the-art real-time analytics systems available to us, including Cloudera Impala [19], Apache Shark [6] and Apache Hive [25]. Since then, OceanRT has been heavily developed and thoroughly optimized. In particular, the new OceanRT employs faster one-sided read/write operations in its RDMA links (described in Section 3.2), a query re-optimization module for query parsing (Section 3.3), direct communication between AQEs in the same node (Section 3.3), and an extended storage scheme that optimizes for spatio-temporal data and queries, while also handling other

Proceedings of the VLDB Endowment, Vol. 7, No. 13

types of data well (Section 3.4). In addition, we have conducted an extensive evaluation of OceanRT using a popular big data benchmark, TPC-DS [17]. The results confirm that OceanRT achieves considerable speedup (often more than an order of magnitude) compared to the current versions of major systems for real-time analytics, especially for complex queries involving joins and subqueries. In the following, Section 2 overviews related work. Section 3 presents the novel designs in OceanRT. Section 4 presents experimental evaluations. Section 5 concludes the paper.

2. RELATED WORK

In the past decade, cloud computing has emerged as the dominant paradigm for performing analytics on big data, since the cloud provides virtually unlimited computing resources on demand. These resources are provided by a massive number of interconnected commodity servers. Hence, to use the cloud effectively, the analytics system should be scalable (i.e., it can handle larger data by using more cloud resources), fault-tolerant (it deals with machine errors and failures gracefully, which are common in a cloud), elastic (it dynamically allocates or releases cloud resources based on the current workload), and efficient (it minimizes resource usage).

Early cloud-based big data analytics systems focus on offline analytics, which processes batches of long-running jobs. Notably, the seminal paper [3] describes Google's MapReduce framework, which provides strong fault tolerance and high parallelism for job processing, while hiding the complexity of the system and exposing a simple programming interface. Soon after the publication of [3], its key ideas were implemented into an open-source system Hadoop [26], which has been widely used. MapReduce, however, is inefficient at handling certain complex, iterative jobs, common in machine learning and graph computation. Addressing this problem, Dryad [10] enables the user to handle a DAG of tasks, and Pregel [13] deals with vertex-centric programming for large graphs. Recently, epiC [11] provides a unified framework for a variety of data types, combining the merits of all afore-mentioned systems. We refer the reader to a comprehensive survey [12] for other related systems.

The successes of offline analytics systems led to attempts to build analytics capabilities on top of them. For instance, earlier versions of Hive [25] translate SQL queries into MapReduce jobs. This approach, however, is not suitable for real-time analytics [15], since offline analytics systems usually incur high query latency. Another methodology is to build MapReduce / RDBMS hybrids. For instance, HadoopDB [1] (which forms the basis of its commercial version, Hadapt) uses relational databases to perform MapReduce tasks. Microsoft PolyBase [4] improves the scalability of SQL Server through "split query processing" [2], which transforms queries into MapReduce jobs. Sailfish [20] accelerates MapReduce by batching disk I/Os. These systems tend to inherit the offline processing designs of MapReduce, and, thus, are not ideal for real-time query processing.

Recently, a plethora of real-time big data analytics systems have been built that do not rely on MapReduce. OceanRT belongs to this category. Most of them, including OceanRT, use SQL as the query language, since SQL is familiar to users and enables interaction with legacy systems [15]. Notably, Google introduced Dremel [14], its real-time analytics tool that focuses on speed (i.e., low query latency) rather than power (support for very complex queries). Cloudera Impala [19] is an open-source real-time SQL processing system over big data inspired by Dremel. Facebook recently replaced Hive with a new system, Presto [22], for real-time analytics. Shark [6] is a real-time SQL processing module built on top of Spark [27], a novel big data processing paradigm based on the idea of resilient distributed datasets (RDDs) [27]. Newer versions of Hive, with a set of optimizations collectively called the "stinger initiative", can also run on faster, non-MapReduce data processing frameworks, e.g., Tez (http://tez.incubator.apache.org), which is inspired by Dryad [10]. Proprietary systems include Amazon Redshift (http://aws.amazon.com/redshift), which, like Dremel, has few technical details publicly revealed.

Finally, another line of research concerns real-time, continuous analytics on fast data streams. Popular open source systems in this category include Storm (https://storm.incubator.apache.org/) and S4 (http://incubator.apache.org/s4/), and Spark Streaming [28]. TimeStream [18] addresses efficient failure recovery in operator-network streaming systems. Resa [24] provides enhanced elasticity, and applies to a wider range of analytics such as clustering [29]. These systems are orthogonal to OceanRT, since we focus on interactive analytics in a data warehouse, rather than continuous analytics over streams.

3. OCEANRT

Section 3.1 overviews the architecture of OceanRT. Sections 3.2-3.4 present the novel networking, computing units, and storage scheme of OceanRT, respectively. In each subsection, we omit the details already presented in the preliminary version [30], and focus on the improvements compared to [30].

3.1 Overview

Similar to most cloud-based systems, an OceanRT cluster consists of interconnected commodity servers. As explained in Section 3.2, although OceanRT employs RDMA technology, we chose a software implementation rather than a hardware one. Hence, the network infrastructure in OceanRT can be simply commodity Ethernet (e.g., 1 GBits or 10 GBits) that is common in today's cloud infrastructure. Similar to existing real-time analytics systems, the user interacts with OceanRT through SQL.

Figure 1 shows the shared-nothing architecture of OceanRT with *n* computing nodes connected through software RDMA links. Each node runs a Parsing Engine (PE), as well as one or more Access-Query Engines (AQEs). Note that unlike the architecture in the preliminary version of OceanRT [30], the AQEs in the same node are connected through lightweight *inter-AQE links*, detailed in Section 3.3. All nodes in OceanRT are symmetric; in other words, there is no distinction among nodes such as master / slaves. Similar to parsers and query executors in a traditional RDBMS, the PE parses incoming SQL queries or sub-queries into execution plans, whereas the AQE is responsible for accessing the data and performing operations in the query plan.

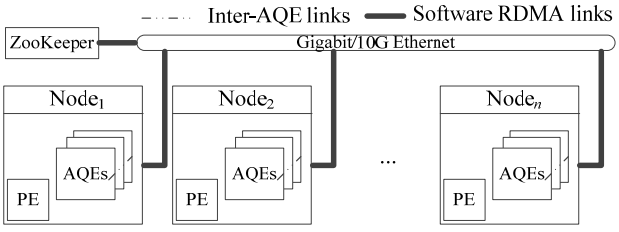


Figure 1: Computing Architecture of OceanRT

OceanRT employs Zookeeper [8] to manage the states of all nodes in the cluster, such as total amount of available memory, CPU utilization percentage, network usage, etc. We omit the details of node/AQE states as they remain largely unchanged from the

preliminary version [30]. AQEs in OceanRT read data from existing cloud-based storage systems, such as HDFS [26], HBase [7], etc. Meanwhile, OceanRT includes a novel storage scheme, explained in Section 3.4, which directs how data is organized in the underlying storage system.

Since all nodes are symmetric, when the user issues a query, one node serves as the gateway that parses the query, coordinates with other nodes to execute it, collects (partial) query results, and returns the final results to the user. In our current implementation, each user connects to a random node upon startup, which serves as the gateway for all queries of this user. Alternatively, to ensure load balancing, we could route each query to the currently least busy node, according to node statuses maintained by Zookeeper. In the current prototype, we observed that the gateway incurs negligible overhead; so, the choice of the gateway node might not be critical. We detail query execution further in Section 3.3. Finally, we mention that the OceanRT architecture does not complicate fault recovery, except for the fact that when one node crashes, all AQEs therein are down. Currently, OceanRT's fault recovery mechanism is similar to that in Impala [19].

3.2 RDMA-Based Networking

Since the cloud comprises a massive number of nodes connected with limited bandwidth, communication between these nodes always incurs a non-trivial cost. This cost is especially prominent in real-time analytics, for two reasons. First, queries in such applications are usually exploratory, and do not involve costly computations; hence, networking overhead takes a larger share of the overall costs. Second, in order to satisfy the real-time response constraint, queries may need to be executed on a large number of nodes to obtain high parallelism, leading to higher communication expenses overall.

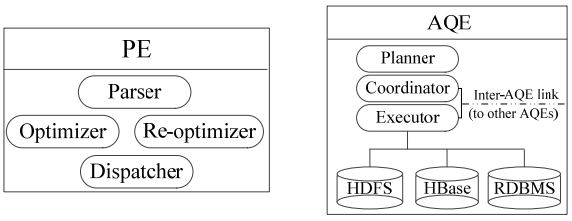
OceanRT tackles these problems by using RDMA extensively instead of traditional socket-based communication methods between nodes. RDMA accelerates network transmissions in two ways. First, RDMA enables zero-copy transmission, meaning that data is directly transmitted from memory pages in the source node to the pages in the destination node, without any intermediate copying between the application memory, the operating system kernel memory, and the socket buffer as in socket-based networking. Second, a hardware implementation of RDMA, common in supercomputers, involves specialized network adaptors with computing capabilities, which perform data transmissions without consuming CPU cycles. However, relying on specialized hardware is also a major disadvantage for applying RDMA in a cloud, since equipping all nodes with RDMA hardware currently requires considerable investment, which may become obsolete soon as the technology advances.

To avoid relying on specialized hardware, OceanRT employs a software implementation of RDMA, namely SoftiWarp [23]. Although network-related operations are still performed by the CPU, SoftiWarp avoids 4 memory copying operations for each data transfer [23], which, in today's increasingly common 10Gbits Ethernet, can cause significant latencies. In particular, OceanRT makes extensive use of the one-sided remote read/write operations of RDMA. For instance, in one-sided writing, before any transmission occurs, the node that will receive data pins a number of RAM pages, which are guaranteed not to be swapped to disk by the OS. After that, a sending node can directly "write" content to these pinned RAM pages in the receiving node with negligible overhead. When the writing is done, SoftiWarp notifies the receiving node, which then reads its pinned memory locally. One-sided reading is symmetric, with the sender pinning RAM pages. Note that the need to pin RAM pages could be a drawback for RDMA when applied in a cloud, since the pinned pages cannot be used for other purposes, leading to reduced flexibility. This problem is alleviated in software RDMA, since memory pages can be unpinned (i.e., swappable to disk) once there is no more transmission [23].

In practice, we found that RDMA accelerates OceanRT mainly in two operations characterized by large message sizes: remote data retrieval and the transmission of intermediate query results. Remote data retrieval occurs, for example, in queries with joins, where a node containing data from one relation must retrieve data from the other relation. Transmission of intermediate results is involved when the query is decomposed into sub-queries, whose results must be subsequently reassembled. For queries that do not involve joins or subqueries, RDMA improves performance by reducing the overhead of the underlying cloud file system. For example, communication costs between a node accessing a large number of blocks from HDFS and the HDFS name node can be significantly reduced using RDMA one-sided reads.

3.3 PE and AQE

Figure 2 shows the internal compositions of the PE and the AQE in OceanRT. Recall from Section 3.1 that a PE parses a SQL query into an execution plan, and multiple AQEs perform the operations in the plan. As shown in Figure 2a, the PE contains four components: a parser, an optimizer, a re-optimizer, and a dispatcher. Specifically, OceanRT invokes the PE in two different occasions; in both uses, the cost of the PE is negligible; hence, we place only one PE per node. First, when a user issues a SQL query to its gateway node, the latter's PE parses the query into an execution plan, optimizes it, decomposes it into multiple subplans, and dispatches each subplan to an AQE for execution. This use of the PE had been implemented in the preliminary version of OceanRT, and we refer the reader to Ref. [30] for details in the query parsing, optimizing, and subplan dispatching steps.



(a) Parsing Engine (b) Access-Query Engine

Figure 2: PE and AQE in OceanRT

The second use of the PE is that when an AQE receives a query subplan, the PE of its corresponding node re-optimizes the subplan, before the AQE executes it. This is a new optimization developed since the preliminary version [30]. The rationale for this step is that the node where the AQE resides may contain more detailed statistics for the data stored locally, which helps optimize the plan further. Instead of using PostgreSQL components as in the parsing and optimizing steps, this re-optimization step is performed by a customized optimizer built from scratch using LLVM[1], which incorporates optimization logic designed specifically for this step.

[1] llvm.org.

An AQE executes the operators in its query subplan, and generates (intermediate) results. As illustrated in Figure 2b, an AQE contains a planner, a coordinator, and an executor, and can read/write data stored in HDFS, HBase, or a relational database. As their respective names suggest, the planner determines the appropriate algorithm for executing the assigned task; the coordinator coordinates with other AQEs to retrieve data (e.g., when processing a join) or partial/subquery results; the executor performs the actions of the logical operator. We omit further details of these components for brevity, which can be found in [30].

Since different AQEs inside a node operate independently, they need to communicate in order to perform certain operations, such as joins and passing sub-query results. In the preliminary version of OceanRT [30], we used RDMA for this purpose, which, though faster than traditional socket-based transmissions, still incurs considerable overhead, such as pinning/unpinning RAM pages, sending acks and checking for errors. We eliminate this overhead by replacing RDMA with a novel inter-AQE link, which resembles an inter-process communication protocol for transmitting data between AQEs (which are run as different processes) in the same node. In particular, sending a piece of information through the inter-AQE link involves little more than a simple memory copy.

Each physical node runs multiple AQEs, depending on its hardware configuration, which obtains a higher degree of parallelism compared to existing systems in which every node is a single processing unit. For instance, consider a node with two hard drives and two CPU cores. By dividing the node into two AQEs, each with a hard drive and a CPU core, the node is able to perform two independent data retrieval and/or processing tasks simultaneously. As our experiments demonstrate, setting the number of AQEs slightly higher than the CPU-and-disk configuration suggests often improves overall performance, e.g., when one AQE is busy reading data from disk, another may perform CPU-intensive computations, and yet another retrieving data from the network. Our current implementation uses a simple heuristic for determining the number of AQEs per node: a (tunable) factor times that number suggested by the CPU-and-disk configuration. A formal analysis for the optimal number of AQEs is left as future work.

3.4 Data Storage

As described in the previous subsection, OceanRT can process data stored in HDFS, HBase, and/or an RDBMS. Currently, data storage in OceanRT ultimately relies on HDFS, since (i) to provide high fault-tolerance, HBase employs HDFS as the underlying file system; and (ii) for the same reason, in OceanRT, the RDBMS (currently PostgreSQL) also stores data files on HDFS. To obtain higher performance, we customized HDFS to better fit OceanRT as well as spatio-temporal data management. Our modifications to HDFS do not compromise the functionalities or fault-tolerance properties of standard HDFS; in other words, OceanRT-optimized HDFS can be directly used in place of HDFS whenever the latter is applicable.

OceanRT involves two major modifications in HDFS. First, as described in [30], we alleviate block fragmentation by grouping blocks of the same file into larger *partitions*, each of which contains M (>1) blocks; meanwhile, we require that each partition be stored completely in at least N nodes. For instance, when M=10 and N=2, each partition contains 10 blocks; these 10 blocks are stored together in at least 2 nodes, each of which is able to scan the entire partition locally without network transmissions. We refer the reader to [30] for further details. From our experiments, we observed that this HDFS-modification alone (i.e., without the second storage optimization, explained soon) does not lead to dramatic performance enhancements. On the other hand, grouping blocks into partitions requires modification to the HDFS source code, which is a limitation to the applicability of OceanRT. For these reasons, we plan to migrate OceanRT storage to stock HDFS in future versions.

Using the two-level partition-block file organization, we further optimize OceanRT storage for spatio-temporal data and queries, as follows. First, we decompose the space into cells, e.g., using an (possibly unbalanced) quad-tree, and assign records in each cell to a partition. Then, in each partition, records are further separated according to their timestamps, and assigned to blocks accordingly. Each partition and block also contains various metadata, including the minimum bounding rectangle (for the spatial attributes) and a time range (for the timestamp) of records therein. Records in each block are then organized using an existing method, e.g., RC-File [9], ORC-File [16], etc.

The grouping of records is accomplished via hashing, i.e., the partition of a record is determined by a hash value of its corresponding spatial cell, and the block of a record is determined by hashing its timestamp. Essentially, this file organization corresponds to an embedded spatio-temporal primary index, whose outer structure is a spatial one; each of its node contains an embedded temporal index. When processing a query with spatial and/or temporal range selections, instead of scanning an entire relation, OceanRT only scans the partitions and blocks whose spatial MBR and temporal range overlap the query.

Besides spatio-temporal data and queries, the novel file organization of OceanRT also applies to other types of data with range selections on multiple attributes, achieving the benefits of a multi-dimensional primary index. In general, for relational and non-spatio-temporal data, the records are first grouped into partitions by their primary key, and then into blocks by an attribute that is frequently involved in range queries or joins. This scheme is used in our experimental evaluations, presented next.

4. EXPERIMENTS

We implemented all core components of OceanRT in C++, plus a foreign data wrapper in Java for interacting with Hadoop, which includes HDFS. The experiments use data and queries from TPC-DS [17], a popular benchmark for comparing big data processing systems, containing 25 tables, 429 columns and 99 query templates. The default data size is set to 1 TByte. Specifically, we use the same query set as a recent experimental evaluation performed by Cloudera [5], which is classified in [5] into 3 categories: interactive, reporting, and deep analytics. Table 1 lists the TPC-DS queries used in our experiments.

Table 1: TPC-DS queries used in the experiments

Type	Queries
Interactive	Q19, Q42, Q52, Q55, Q63, Q65, Q68, Q73, Q98
Reporting	Q3, Q7, Q27, Q43, Q53, Q89
Deep Analytics	Q34, Q46, Q59, Q79, SS_MAX

According to [5], as of mid-2014, the current version of Impala (v1.3.1) outperforms by clear margins the current versions of Presto, Shark, and Stinger Hive-on-Tez, which are currently the most popular open-source real-time big data analytics systems, as described in Section 2. Hence, we consider Impala v1.3.1 as our main competitor. Since Impala's compiler supports only a subset of TPC-DS queries [21], the evaluation of Impala utilizes the Impala TPC-DS Kit from Cloudera for re-writing queries, available at

https://github.com/cloudera/impala-tpcds-kit. In addition, we also compare OceanRT against Hive-on-Tez v0.12, and we have published the query re-writing toolkit for Hive at https://github.com/simonzhangsm/hive-testbench. The version of Hadoop used in the experiments is 2.3.0. These versions of Impala, Hive and Hadoop are contained in CDH 5.0.2.

All experiments were run in a private cluster consisting of 10 nodes, each equipped with 2 Intel Xeon 1.9GHz 6-core CPUs, 16 GBytes of memory, and 2 1-TByte hard drives. All nodes run on 64-bit Ubuntu Linux 12.04 LTS. By default, there are 4 AQEs in each node. Figure 3 exhibits the network topology of the nodes.

Figure 4 presents the evaluation results of Hive-on-Tez, Impala, and OceanRT on all 20 queries in Table 1, with the dataset size fixed to 1 TBytes. The performance trends in our results are consistent with [5], with Impala considerably faster than Hive-on-Tez. On all queries, OceanRT consistently and significantly outperforms Impala, often by more than an order of magnitude. The performance advantage of OceanRT is particularly pronounced for the interactive queries. For example, on Q98, OceanRT achieves 13x and 106x sppedup compared to Impala and Hive-on-Tez, respectively. It is worth mentioning that OceanRT finishes processing every query within 7 seconds, which is acceptable waiting time for most interactive applications.

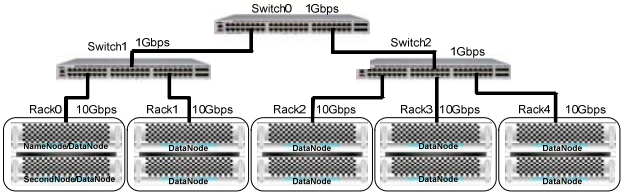


Figure 3: Network topology of the nodes in the experiments

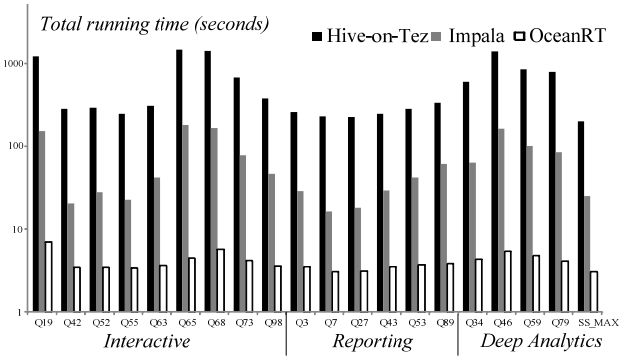


Figure 4: Evaluation on all queries using 1TB data

Figure 5 plots the total running time as a function of the dataset size for two sample interactive queries, Q52 and Q68. Results with other interactive queries are omitted since they lead to similar conclusions. The running times of all systems increase with the dataset size, as expected. Clearly, OceanRT is the fastest in all settings. The performance gap between OceanRT and Impala/Hive generally expands as the data size grows (note that the running times are shown in log scale).

Figure 6 and Figure 7 repeat the above experiment on sample reporting and deep analytics queries, respectively. OceanRT again wins on all settings, with clear margins that generally grow with the dataset size. Comparing the results on different types of queries, the performance advantage of OceanRT is more pronounced on complex queries such as Q46 (up to 30x faster than Impala) than on simpler ones such as Q52 (up to 8x faster than Impala). This is because complex queries generally involve more network transmissions and data retrievals, leading to more pronounced effects of the RDMA links and the novel storage scheme in OceanRT.

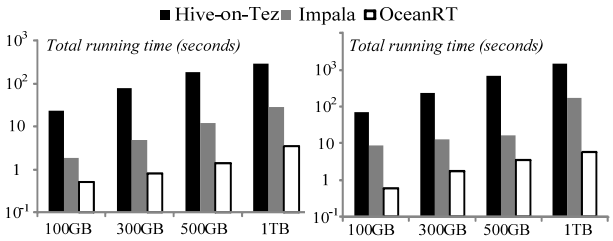


Figure 5: Effect of dataset size on sample interactive queries

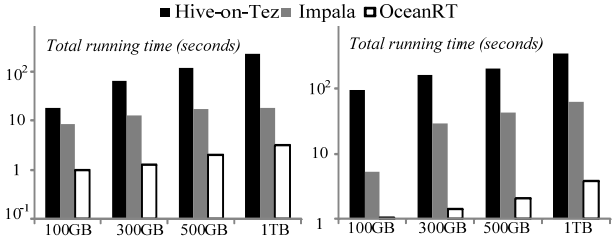


Figure 6: Effect of dataset size on sample reporting queries

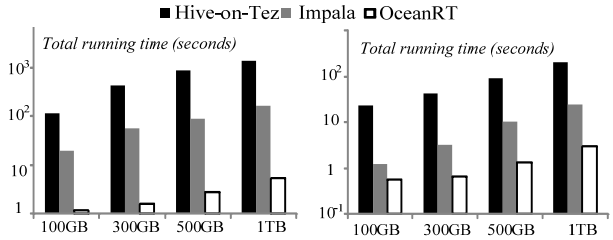


Figure 7: Effect of dataset size on deep analytics queries

Having examined the performance of OceanRT compared to existing systems, we next focus on its internal parameters. Figure 8 shows the impact of the number of AQEs in each physical node on the overall performance of OceanRT, using the 6 sample queries studied in Figures 5-7. Recall that each node has two CPUs with 6 cores each, 2 hard drives, and 16 GBytes of RAM. Hence, splitting each node into 2 AQEs would lead to each having 1 CPU and 1 hard drive; a higher number of AQEs forces different AQEs to share resources, especially the hard drive. According to Figure 8, having multiple AQEs leads to significantly better performance than treating each node as a single computing unit, which is the case in most existing systems. Meanwhile, the query response time of OceanRT continues to drop as the number of AQEs increases beyond 2, indicating that the benefit of increased parallelism outweighs the drawback of resource competition between AQEs. The performance of OceanRT stabilizes when the number of AQEs reaches around 8; after that, adding more AQEs adversely affects performance. These observations suggest there exists an optimal number of AQEs per node that cannot be determined by the amount of resources in a straightforward way. Hence, further investigation into this topic is an interesting direction for future work.

Summarizing the experiments, OceanRT outperforms existing systems in every single setting tested, with a performance gap that

grows with the dataset size as well as the complexity of the query. For 1TB data, OceanRT running on a small cluster of 10 nodes finishes every query within 7 seconds. Finally, the number of AQEs needs to be carefully tuned to maximize performance.

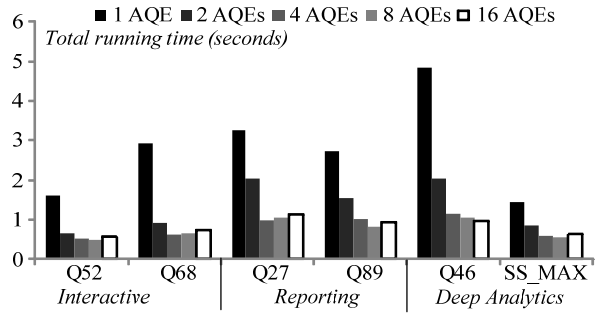


Figure 8: Effect of different number of AQEs per node

5. CONCLUSION

We present OceanRT, an interactive analytics platform for big data, especially spatio-temporal data and queries. OceanRT achieves higher performance compared to existing systems, through three innovative system designs: the use of RDMA links between nodes, the novel architecture involving multiple AQEs per node, and a storage scheme that reduces file fragmentation and serves the purpose of a multi-dimensional primary index. Regarding future work, an interesting direction is to derive a formal performance model for OceanRT, which can help determine the best number of AQEs per node, a critical system parameter. Meanwhile, the RDMA links may be improved by minimizing pinning / unpinning of memory pages. Finally, we intend to optimize the storage scheme further to reduce the number of remote block retrievals.

6. ACKNOWLEDGMENTS

Yin Yang and Marianne Winslett are supported by A*STAR under the Human Centric Cyber Systems (HCCS) Program.

7. REFERENCES

[1] Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Silberschaz, A., Rasin, A. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. *VLDB*, 2009.

[2] Bajda-Pawlikowski, K., Abadi, D., Silberschatz, A., Paulson, E. Efficient Processing of Data Warehousing Queries in a Split Execution Environment. *SIGMOD*, 2011.

[3] Dean, J., Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. *OSDI*, 2004.

[4] DeWitt, D. J., Halverson, A., Nehme, R., Shakar, S., Aguilar-Saborit, J., Avanes, A., Flasza, M., Gramling, J. Split Query Processing in Polybase. *SIGMOD*, 2013.

[5] Erickson, J., Kornacker, M., Kumar, D. New SQL Choices in the Apache Hadoop Ecosystem: Why Impala Continues to Lead. *http://blog.cloudera.com/blog/2014/05/new-sql-choices-in-the-apache-hadoop-ecosystem-why-impala-continues-to-lead/*.

[6] Engle, C., Lupher, A., Xin, R., Zaharia, M., Franklin, M., Shenker, S., Stoica, I. Shark: Fast Data Analysis Using Coarse-Grained Distributed Memory. *SIGMOD*, 2012, demo.

[7] George, L. *HBase: The Definitive Guide – Random Access to Your Planet-Size Data*. O'Reilly, 2011.

[8] Hunt, P., Konar, M., Junqueira, F. P., Reed, B. ZooKeeper: Wait-Free Coordination for Internet-Scale Systems. *USENIX ATC*, 2010.

[9] He, Y., Lee, R., Huan, Y., Shao, Z., Jain, N., Zhang, X., Xu, Z. RCFile: A Fast and Space-Efficient Data Placement Structure in MapReduce-Based Warehouse Systems. *ICDE*, 2011.

[10] Isard, M., Budiu, M., Yuan, Y., Birrell, A., Fetterly, D. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. *ACM Eurosys*, 2007.

[11] Jiang, D., Chen, G., Ooi, B. C., Tan, K. -L., Wu, S. epiC: an Extensible and Scalable System for Processing Big Data. *PVLDB*, 7(7), 2014.

[12] Li, F., Ooi, B. C. Ozsu, T., Wu, S. Distributed Data Management Using MapReduce. *ACM Computing Survey*, 46(3), 2014.

[13] Malewicz, G., Austern, M. H., Bik, A. J. C., Dehnert, J. C., Horn, I., Leiser, N., Czajkowski, G. Pregel: a System for Large-Scale Graph Processing. *SIGMOD*, 2010.

[14] Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., Vassilakis, T. Dremel: Interactive Analysis of Web-Scale Datasets. *VLDB*, 2010.

[15] Olson, M. Impala v Hive. *http://vision.cloudera.com/impala-v-hive/*.

[16] Propopp, C. ORC: An Intelligent Big Data File Format for Hadoop and Hive. *http://www.semantikoz.com/blog/orc-intelligent-big-data-file-format-hadoop-hive/*.

[17] Poess, M., Nambiar, R. O., Walrath, D. Why You Should Run TPC-DS: A Workload Analysis. *VLDB*, 2007.

[18] Qian, Z., He, Y., Su, C., Wu, Z., Zhu, H., Zhang, T., Zhou, L., Yu, Y., Zhang, Z. TimeStream: Reliable Stream Computation in the Cloud. *ACM EuroSys*, 2013.

[19] Russell, J. *Cloudera Impala*. O'Reilly, 2013.

[20] Rao, S., Ramakrishnan, R., Silberstein, A., Ovsiannikov, M., Reeves, D. Sailfish: A Framework for Large Scale Data Processing. *SoCC*, 2012.

[21] Soliman, M. A., Antova, L., Raghavan, V., El-Helw, A., Gu, Z., Shen, E., Caragea, G. C., Garcia-Alvarado, C., Rahman, F., Petropoulos, M., Waas, F., Narayanan, S., Krikellas, K., Baldwin, R. Orca: A Modular Query Optimizer Architecture for Big Data. *SIGMOD*, 2014.

[22] Traverso, M. Presto: Interacting with petabytes of data at Facebook. *https://www.facebook.com/notes/facebook-engineering/presto-interacting-with-petabytes-of-data-at-facebook/10151786197628920*

[23] Trivedi, A., Metzler, B., Stuedi, P. A Case for RDMA in Clouds: Turning Supercomputer Networking into Commodity. *ACM APSys*, 2011.

[24] Tan, T, Ma, R., Winslett, M., Yang, Y., Yong, Y., Zang, Z. Realtime Elastic Streaming Analytics in the Cloud. *SIGMOD*, 2013, poster.

[25] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R. Hive – A Warehousing Solution Over a Map-Reduce Framework. *VLDB*, 2009.

[26] White, T. *Hadoop: The Definitive Guide – MapReduce for the Cloud*. O'Reilly, 2009.

[27] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M., Shenker, S., Stoica, I. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. *NSDI*, 2012.

[28] Zaharia, M., Das, T., Li, H., Shenker, S., Stoica, I. Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters. *ACM SOSP*, 2013.

[29] Zhang, Z., Shu, H., Chong, Z., Lu, H., Yang, Y. C-Cube: Elastic Continuous Clustering in the Cloud. *ICDE*, 2013.

[30] Zhang, S., Yang, Y., Fan, W., Lan, L., Yuan, M. OceanRT: Real-Time Analytics over Large Temporal Data. *SIGMOD*, 2014, demo.