OceanRT: Real-Time Analytics over Large Temporal Data

Shiming Zhang* Yin Yang# Wei Fan* Liang Lan* Mingxuan Yuan*
*Noah's Ark Lab, Huawei, Science Park, Shatin, Hong Kong
{simon.zhangsm, david.fanwei, lan.liang, yuan.mingxuan}@huawei.com
Advanced Digital Sciences Center, Singapore
yin.yang@adsc.com.sg

ABSTRACT

We demonstrate OceanRT, a novel cloud-based infrastructure that performs online analytics in real time, over large-scale temporal data such as call logs from a telecommunication company. Apart from proprietary systems for which few details have been revealed, most existing big-data analytics systems are built on top of an offline, MapReduce-style infrastructure, which inherently limits their efficiency. In contrast, OceanRT employs a novel computing architecture consisting of interconnected Access Query Engines (AQEs), as well as a new storage scheme that ensures data locality and fast access for temporal data. Our preliminary evaluation shows that OceanRT can be up to 10x faster than Impala [10], 12x faster than Shark [5], and 200x faster than Hive [13]. The demo will show how OceanRT manages a real call log dataset (around 5TB per day) from a large mobile network operator in China. Besides presenting the processing of a few preset queries, we also allow the audience to issue ad hoc HiveQL [13] queries, watch how OceanRT answers them, and compare the speed of OceanRT with its competitors.

Categories and Subject Descriptors

H.2 [**Database Management**]: Systems

General Terms

Design, Management, Performance

1. INTRODUCTION

We have entered the big data era, in which organizations routinely collect terabytes or even petabytes of data. For instance, one of Huawei's client, a large mobile network operator in China, accumulates over 5 TB of call logs each day. The capability to gain useful knowledge from such big data is thus crucial to the success of today's organizations. In the past much work has been done to develop the capabilities to perform offline analytics, which involve complex tasks that take hours or days to finish, even on a modern cloud platform. For example, MapReduce [3] harnesses the computing power of interconnected commodity servers, provides strong fault tolerance and high parallelism for job processing, while hiding the complexity of the system and exposing a simple interface to users. Recently, considerable attention has been shifted to answering analytics queries online (also referred to as *interactive* or *real-time* analytics), which outputs results within seconds once the user issues a query. Such capabilities are an important complement to offline analytics; in particular, they enable users to quickly explore and obtain initial insights of the big data. Since the user often waits online for the results, speed is key in online analytics. Meanwhile, in such applications a simple programming language such SQL is often preferred, since it lets the user focus on the logic of the analytics, rather than implementation details in, e.g., a MapReduce program.

Early attempts build analytics capabilities on top of a MapReduce system, e.g., Hadoop [14]. Notably, Hive [13] translates SQL queries into MapReduce jobs, and subsequently executes them using Hadoop. YSmart [7], another SQL-to-MapReduce translator, provides sophisticated optimizations for complex queries. This approach, however, is not suitable for online analytics, since MapReduce is designed for offline tasks, and, thus, involves high overhead for starting and executing each job. Another methodology is to build MapReduce/RDBMS hybrids. For instance, HadoopDB [1] (commercialized as Hadapt [2]) uses relational databases to perform MapReduce tasks. PolyBase [4] improves the scalability of SQL Server using a novel "split query processing" method that transforms queries into MapReduce jobs. Finally, Sailfish [9] improves MapReduce performance 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, following Google's publication of Dremel [8] and F1 [12, 11], there has been increasing interests in building real-time query processing tools that answer queries directly, without invoking MapReduce. For example, Cloudera Impala [10] processes SQL queries over data stored in HDFS, the file system of Hadoop, and/or in HBase[1], a "NoSQL" data management system that features a high level of schema flexibility at the expense of limited query processing capabilities compared to an RDBMS. Shark [5] is the online query processing module built on top of Spark[15], a novel in-memory big data processing platform. Proprietary systems include Amazon Redshift[2], which, like Google's Dremel and F1, has few technical details revealed.

Our goal is to perform online, real-time analytics over big temporal data, such as call logs from a large mobile network operator. Queries on such data often contain temporal operations. We observe that existing systems have two major limitations that render them unsuitable for our cause. First, as we show in Section 2,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.
Copyright 2014 ACM 978-1-4503-2376-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2588555.2594513.

[1]http://hbase.apache.org
[2]http://aws.amzaon.com/redshift

the current generation of existing systems cannot fully utilize the parallel computing capabilities of modern servers, which typically have multiple CPU cores and hard drives. While it is possible to subdivide each server into virtual machines, doing so incurs high overhead, e.g., communications between virtual machines has to go through network sockets. Second and more importantly, existing systems are not optimized for temporal data. Hence, we have been building OceanRT (pronounced "ocean art"), a novel online analytics engine for large temporal data. Instead of reusing components from a MapReduce-like system, OceanRT redesigns the computing architecture and storage scheme to obtain a higher level of parallelism and improved data locality, and to manage temporal data more efficiently. We have implemented a prototype of OceanRT, and evaluated it using large real data and queries. The results indicate that OceanRT can obtain up to 10x speedup compared to the current state-of-the-art, especially for complex queries involving joins and subqueries. We thus plan to demonstrate OceanRT to SIGMOD attendees with real data and use cases.

In the following, Section 2 describes the OceanRT system. Section 3 presents preliminary evaluation results comparing OceanRT with existing systems on real data. Section 4 describes our plan for the demo. Section 5 contains concluding remarks.

2. SYSTEM OVERVIEW

OceanRT includes both a novel computing architecture and a new storage scheme for performing analytics in real time over temporal data. Sections 2.1 and 2.2 present the computing architecture and the storage scheme of OceanRT, respectively.

2.1 Computing Architecture

An OceanRT cluster consists of multiple interconnected commodity servers. We employ Zookeeper[3] to manage the states of the nodes. Similar to Impala, the user can submit a HiveQL query (a dialect of SQL used in Hive [13]) to any node in OceanRT; the node receiving the query is responsible for parsing the query, dispatching (parts of) the query to relevant nodes for execution, collecting (partial) query results, and returning the results to the user. To do this, each node runs a Parsing Engine (PE), shown in Figure 1(a). The PE contains (i) a parser, which parses incoming SQL queries into execution plans, (ii) an optimizer, which optimizes the plans, e.g., by re-ordering joins and (iii) a dispatcher, which routes the query to nodes that stores relevant data. OceanRT employs PostgreSQL[4] parser and optimizer modules in its PE, which leverages the time-proven query optimization techniques in PostgreSQL.

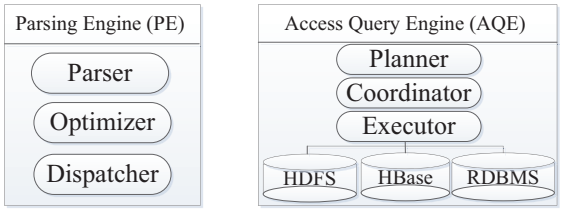

Figure 1: PE and AQE in OceanRT

After generating the query execution plan, the dispatcher module of the PE assigns each logical operator to an Access Query Engine (AQE), which executes the operator and generates (intermediate) results. The dispatching is performed according to the AQE state information (e.g., busy/idle) maintained by Zookeeper, as well as the network distance between an AQE and the data required by the logical operator (e.g., AQEs with local data are preferred). As we detail soon, each node runs multiple AQEs, depending on its hardware configuration. As illustrated in Figure 1(b), an AQE contains a planner, a coordinator, and an executor. It can read/write data stored in HDFS, HBase, or a relational database such as PostgreSQL. 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 action specified in the logical operator.

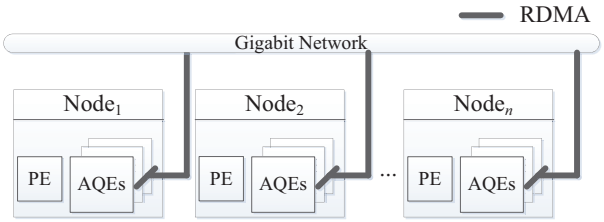

Figure 2: Architecture of OceanRT

So far, our designs for the PE and AQE are fairly standard for an online analytics engine, e.g., they resemble the design of the Impala daemon, albeit more refined with the help of PostgreSQL. The main novelty in the architectural design of OceanRT lies in two aspects: (i) each physical nodes runs multiple AQEs, according to its hardware configuration, and (ii) all AQEs are connected through Remote Direct Memory Access (RDMA) [6]. Figure 2 shows the architecture of OceanRT. Having multiple AQEs inside each node enables OceanRT to achieve a higher degree of parallelism, compared to existing real-time analytics 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 computation resources 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. In practice, a higher number of AQEs may improve 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. Determining the appropriate number of AQEs per node, and scheduling them properly to maximize performance are interesting research topics of their own.

All AQEs are connected through an RDMA link, which performs data transmission directly between two memory buffers, without consuming CPU cycles or copying data to/from kernel memory. This implies (i) that data transfer between different AQEs within the same node incurs negligible overhead, as the data are simply copied from one memory location to another, and (ii) data exchange between AQEs in different nodes are also faster than existing systems. Our current prototype of OceanRT applies a software implementation of RDMA contained in Hadoop-RDMA [6]. Hardware RDMA promises even better performance. Note that effective use of RDMA is non-trivial as it complicates the memory management of an AQE, such as a page that is currently being transferred to another AQE under RDMA must not be swapped to disk.

2.2 Storage Scheme

As described in Section 2.1, OceanRT can process data stored in HDFS, HBase, and/or a relational DBMS. In order to provide high fault-tolerance, HBase uses HDFS as the underlying file system. For the same reason, in our current implementation of OceanRT,

[3]http://zookeeper.apache.org
[4]http://www.postgresql.org

the RDBMS (in particular, PostgreSQL) also employs HDFS to store data files. Hence, data storage in OceanRT ultimately relies on HDFS. To further improve the performance of OceanRT, we modified HDFS to better fit the computing architecture of OceanRT as well as temporal data management, leading to a novel storage scheme. It is worth mentioning that our modifications to HDFS do not compromise the functionalities of stock HDFS; in other words, OceanRT-optimized HDFS can be directly used in place of HDFS whenever the latter is applicable.

In a nutshell, HDFS achieves fault tolerance as follows. Each data file is partitioned into blocks of fixed size, e.g., 128MBytes. Each block is replicated on multiple servers (3 by default). Specifically, when appending a new block to a file, stock HDFS stores one copy of the block at the node performing the write operation, another copy at a random node at the same rack as the first copy, and yet another copy at a random node not in the same rack [14]. Accordingly, after writing the entire file to HDFS, the node performing the write operation stores a complete copy of the file; the other two copies of the same file, however, are fragmented, among nodes within the same rack and nodes not in the same rack, respectively. Such file fragmentation adversely affects the performance of OceanRT. For instance, consider the situation that the query requires scanning multiple blocks of a file, and the node storing the complete file is busy. The PE then dispatches the scan to an AQE that only stores parts of the file, which must request for blocks from other nodes, leading to high latency due to network transmissions.

To alleviate file fragmentation, OceanRT organizes blocks of a file into larger partitions. Each partition contains M blocks; meanwhile, each partition is 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. M and N are system parameters whose best values depend upon the application. Higher values of M and/or N lead to a lower level of fragmentation, but also increased risk for workload imbalance, and vice versa. Hence, a larger M and/or N is more suitable for applications that involve the scanning of large portions of files. Note that organizing blocks into larger partitions is different from having large block sizes; the latter approach leads to increased network transmissions as a block is a basic unit of reading, as well as limited parallelism. Meanwhile, since the smallest unit of storage is still a block, introducing partitions does not compromise existing functionalities of HDFS. Hence, OceanRT-optimized-HDFS is backward compatible with stock HDFS.

Finally, using the new partition-block file organization, we optimize OceanRT storage for temporal data and queries, as follows. Each block of a temporal data file is assigned a hash value computed based on the temporal attributes of the records contained in the block. Then, blocks are assigned to partitions according to their hash values. Thus, each partition stores data with similar temporal attributes, achieving the effect of a primary index on the temporal attributes. When processing a query with a temporal range selection, instead of scanning an entire relation, OceanRT only scans the partitions that contain records whose temporal attributes fall within the range. Note that building a traditional primary index is not feasible, as such an index requires sorting the entire file, which is costly for big data; further, since updates are not ordered by time, they may cause frequent re-organizations, which is prohibitively expensive. The hashing scheme in OceanRT does not incur these costs; as a tradeoff, it also retrieves a larger portion of the data compared to a traditional primary index. We omit further details for brevity, and our demo will show the data storage of OceanRT using a real example.

3. PERFORMANCE OF OCEANRT

We use a real mobile call log dataset to evaluate OceanRT. Due to data confidentiality issues, we are not allowed to publish details of the data or its schema. Hence, for the sake of the evaluation, we extracted a random sample from the real data, and transformed the sample to fit an artificial schema. Figure 3 illustrates this schema, which contains 5 tables: Users, Services, Locations, CallLogs, and Apps. Among these, Services, CallLogs and Users contain temporal data.

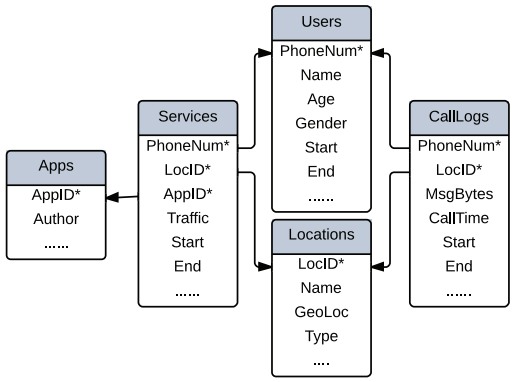

Figure 3: Schema of the dataset used in the evaluations

Figure 4 lists the three queries used in our evaluations, which were composed based on real analytics tasks and the schema shown in Figure 3. Queries Q1 and Q2 represent single-table selections and GROUPBY aggregations, respectively. Q3 is a complex join query involving multiple subqueries, GROUPBY aggregation, sorting and top-k selection. All three queries involve temporal selections, which are common in analytics tasks on the call log dataset.

Q1: Single-table selection

```
SELECT PhoneNum, AppID, LocID
FROM Services
WHERE Traffic > 100M AND Start ≥ DATE('20131001')
AND End ≤ DATE('20131003')
```

Q2: Single-table aggregation

```
SELECT PhoneNum, LocID, SUM(MsgBytes), SUM(CallTime)
FROM CallLogs
WHERE Start ≥ DATE('20131001')  AND End ≤ DATE('20131003')
GROUP BY PhoneNum, LocID
```

Q3: Complex join with subqueries

```
SELECT Name, SUM(MsgBytes) SumMB, SUM(CallTime) SumCT,
    SUM(Traffic) SumTR
FROM Users U
    INNER JOIN CallLogs C ON U.PhoneNum = C.PhoneNum
    INNER JOIN Services S ON C.PhoneNum = S.PhoneNum AND
    C.LogID = S.LocID
WHERE LocID IN
    (SELECT LocID FROM Locations
     WHERE GeoLoc in ('GUANGZHOU', 'SHENZHEN'))
AND AppID IN
    (SELECT AppID FROM Apps WHERE Author='YOUTUBE')
AND S.Start ≥ DATE('20131001')  AND S.End ≤ DATE('20131003')
AND U.Age ≥ 18
GROUP BY U.name ORDER BY SumMB, SumCT, SumTF
LIMIT 1000
```

Figure 4: Queries used in the evaluation

We performed preliminary evaluations on a small cluster of 10 commodity servers connected through a Gigabit Ethernet. Each node is equipped with two Intel i7-3770 CPUs, 16 GB RAM, and two 1TB hard drives. OceanRT assigns 2 AQEs per node. The HDFS block size is fixed to 64MB. Figure 5 summarizes the evaluation results, comparing OceanRT with Hive 0.10, Shark 0.80 and

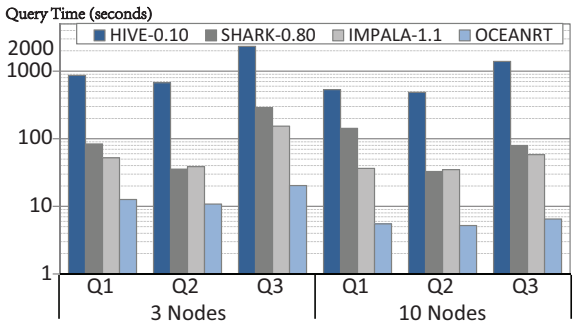

Figure 5: Evaluation results

Impala 1.1. For all three queries, OceanRT is significantly faster than its competitors. The performance advantage of OceanRT is more pronounced with more nodes in the cluster, and for complex queries such as Q3. The performance of Shark and Impala are comparable, with Impala slightly faster than Shark. OceanRT is between 3x (Q2 with 3 nodes) to 10x (Q3 with 10 nodes) faster than Impala, and between 60x to 200x faster than Hive.

4. DEMONSTRATION SCENARIO

The demo will show the system architecture and storage scheme of OceanRT. It will also compare the performance of OceanRT with its competitors using the real call log dataset from a large Chinese mobile network operator, which contains roughly 5 TB (about 4 billion records) per day. A random sample of the data is used in our performance evaluations in Section 3.

Setup. Since the dataset is confidential, we will host it in a private cloud with 100 nodes, and connect to the cloud remotely during the demo. Meanwhile, the demo will only show the performance (in terms of total running time and a breakdown into CPU, I/O, networking costs, etc.) of different systems, not the data records or query answers. OceanRT and its competitors, e.g., Hive, Impala, Shark and possibly others, will be deployed to the private cloud. Besides showing a performance comparison with a set of preset queries, SIGMOD attendees can also issue ad hoc queries to any of the deployed systems, using a command line interface.

Visualization. We will visualize the computing architecture and storage scheme of OceanRT during the demo. Specifically, we will show how the data is stored in the cloud, including statistics on blocks, partitions, and temporal information. Using a number of pre-selected queries, including Q1-Q3 described in Section 3, we will demonstrate how these queries are parsed into logical execution plans, and dispatched to AQEs for execution. We will also show the internals of an AQE when executing a task, such as table scan, join, and GROUPBY/aggregation.

Story line. Consider that a mobile network operator wants to investigate the behaviors of its subscribers in order to launch a new promotion campaign. To do this, it first finds heavy users of the mobile network, and analyzes their phone call patterns (e.g., whether they make calls during the day or at night), web browsing history (e.g., frequently visited websites), usage of value-add services (e.g., customized ringtones), and geo-locations. Using this information, the mobile network operator decides on the selected customers of the campaign, as well as the service package, e.g., browsing a news website will be free of charge at night for customers in an area who subscribe to a value-add service. Since there are numerous possible designs of the campaign, the mobile network operator may issue many queries, and expect each to return results within seconds. Hence, low-latency query processing is key in this application.

5. CONCLUSION

We have been building OceanRT, a real-time data analytics tool for large temporal data. Compared to existing systems, OceanRT employs a novel computing architecture with enhanced parallelism, and a new storage scheme optimized for the new system architecture as well as temporal data and queries. Evaluations using real data and workload show that OceanRT can achieve 10x speedup compared to state-of-the-art systems such as Impala and Shark. The demo will show the performance and internal workings of OceanRT, using real data on a private cloud. Regarding future work, and interesting direction is to incorporate elastic resource provisioning into OceanRT, so that each query only consumes cloud resources (e.g., AQEs) sufficient to meet its quality-of-service requirements [16].

6. REFERENCES

[1] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin. HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads. *PVLDB*, 2(1), 2009.

[2] K. Bajda-Pawlikowski, D. J. Abadi, A. Silberschatz, and E. Paulson. Efficient processing of data warehousing queries in a split execution environment. In *ACM SIGMOD*, 2011.

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

[4] D. J. DeWitt, A. Halverson, R. Nehme, S. Shankar, J. Aguilar-Saborit, A. Avanes, M. Flasza, and J. Gramling. Split query processing in polybase. In *ACM SIGMOD*, 2013.

[5] C. Engle, A. Lupher, R. Xin, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: Fast data analysis using coarse-grained distributed memory. In *ACM SIGMOD*, 2012.

[6] N. S. Islam, M. W. ur Rahman, J. Jose, R. Rajachandrasekar, H. Wang, H. Subramoni, C. Murthy, and D. K. Panda. High performance RDMA-based design of hdfs over infiniband. In *IEEE/ACM SC*, 2012.

[7] R. Lee, T. Luo, Y. Huai, F. Wang, Y. He, and X. Zhang. YSmart: Yet another SQL-to-MapReduce translator. In *IEEE ICDCS*, 2011.

[8] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: Interactive analysis of web-scale datasets. *PVLDB*, 3(1-2), 2010.

[9] S. Rao, R. Ramakrishnan, A. Silberstein, M. Ovsiannikov, and D. Reeves. Sailfish: A framework for large scale data processing. In *ACM SOCC*, 2012.

[10] J. Russell. Cloudera impala: Real-time queries in apache hadoop. In *OŕEilly Strata Conference*, 2013.

[11] J. Shute, M. Oancea, S. Ellner, B. Handy, E. Rollins, B. Samwel, R. Vingralek, C. Whipkey, X. Chen, B. Jegerlehner, K. Littlefield, and P. Tong. F1: The fault-tolerant distributed RDBMS supporting Google's ad business. In *ACM SIGMOD*, 2012.

[12] J. Shute, R. Vingralek, B. Samwel, B. Handy, C. Whipkey, E. Rollins, M. Oancea, K. LittleiňĄeld, D. Menestrina, S. Ellner, J. Cieslewicz, I. Rae, T. Stancescu, and H. Apte. F1: A distributed SQL database that scales. In *VLDB*, 2013.

[13] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: A warehousing solution over a map-reduce framework. *PVLDB*, 2(2), 2009.

[14] T. White. *Hadoop: The Definitive Guide, MapReduce for the Cloud*. O'Reilly Media, 2nd edition, 2009.

[15] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *USENIX NSDI*, 2012.

[16] Z. Zhang, R. Ma, J. Ding, and Y. Yang. Abacus: An auction-based approach to cloud service differentiation. In *IEEE IC2E*, 2013.