Cost-Effective Data Analytics across Multiple Cloud Regions

Junyi Shu, Xin Jin, Yun Ma, Xuanzhe Liu, Gang Huang
Peking University

ABSTRACT

We propose a cloud-native data analytics engine for processing data stored among geographically distributed cloud regions with reduced cost. A job is split into subtasks and placed across regions based on factors including prices of compute resources and data transmission. We present its architecture which leverages existing cloud infrastructures and discuss major challenges of its system design. Preliminary experiments show that the cost is reduced by 15.1% for a decision support query on a four-region public cloud setup.

CCS CONCEPTS

• **Networks** → **Cloud computing**; • **Computer systems organization** → **Distributed architectures**.

KEYWORDS

data analytics, cost optimization, job scheduling, multi-cloud

ACM Reference Format:
Junyi Shu, Xin Jin, Yun Ma, Xuanzhe Liu, Gang Huang. 2021. Cost-Effective Data Analytics across Multiple Cloud Regions. In *SIGCOMM '21 Poster and Demo Sessions (SIGCOMM '21 Demos and Posters), August 23–27, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3472716.3472842

1 INTRODUCTION

Cloud computing has been widely adopted by industry over the past decade. Services deployed in data centers which are closer to customers can achieve lower latency on the client side [7]. Therefore, major cloud providers make services available in multiple regions all over the world [4, 10].

There are already a few enterprises running business worldwide and using cloud resources from multiple regions [11]. Understanding operational statistics is crucial for optimizing systems and making business decisions. Cloud providers have provided tools to aggregate log data for DevOps purposes [3]. Sometimes, business questions need to be answered based on data from multiple regions. For example, a cross-border E-commerce platform may have supplier information stored in Asia and customer information stored in the United States that are both needed to find out customers' preferences over certain characteristics of suppliers.

However, running analytics queries on multi-region data is costly and inefficient. Users of data analytics platforms, such as Snowflake [15], have to replicate necessary data to a single region before using it, but transferring infrequently used data to another region incurs extra transmission and storage cost. Furthermore, cross-continent data transmission is also time-consuming. It takes 5 minutes on average to transfer a 1GB file from AWS Cape Town region to Sydney region.

There have been existing works studying cost optimization of cloud computing. Many of these works aim to optimize cost for a certain type of cloud resources, such as VM instances [8, 9]. There are also studies on job placement across the network which take bandwidth and latency into account [6, 12]. Job completion time (JCT) is another optimization goal for geo-distributed data analytics [13, 17]. A common approach for such problems is to build a mathematical model with an objective function and a set of constraints, and apply a linear/non-linear programming solver to find the optimal solution.

In contrast, we face a new set of problems when orchestrating data analytics jobs across cloud regions, which requires a practical solution to minimize cost. **(a) Different and dynamic prices of compute and storage resources**: an EC2 m5.2xlarge instance costs about 59% more in São Paulo region than in N. Virginia region, and prices of spot instances are changing constantly. **(b) Different and asymmetric prices of data transmission**: prices vary for each region, and data transfer from continents such as Africa and South America costs up to 13× more compared to that from North America for AWS. **(c) Heterogeneous, dynamic and asymmetric network bandwidth**: the time to send data is 10× more when crossing continents, and fluctuates all the time. **(d) Long running jobs with complex workflows**: it is common for analytics queries to take hours to run, and there is usually more than one potential execution plan. **(e) Dynamic datasets**: the size and availability of a dataset may change at any point of time. Many assumptions of previous works do not stand when these new problems arise.

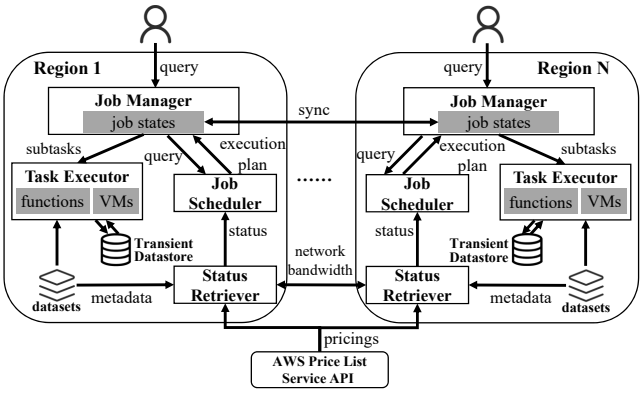

Figure 1: System Architecture

SIGCOMM '21 Demos and Posters, August 23–27, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8629-6/21/08…$15.00
https://doi.org/10.1145/3472716.3472842

2 SYSTEM ARCHITECTURE

Our idea is to continuously monitor all factors affecting cost and performance of cross-region analytics jobs and derive a cost-effective execution plan bounded by a given JCT requirement. Each region is treated equally so there is no single point of failure. Our solution leverages existing cloud offerings such as spot VMs and serverless functions to achieve high scalability. Figure 1 shows the building blocks of it.

Status retriever. There is a status retriever in each region, which takes control of collecting and synchronizing mandatory information for job scheduling. It collects (a) pricing from respective cloud vendors through APIs, (b) network bandwidth by periodically communicating to other regions, and (c) metadata of datasets in its region. The most crucial parameters that it collects are prices of data transfer between regions and prices of compute resources in each region. Bandwidths are also helpful as we may want to set a limit on job completion time. Status retrievers in all regions have to synchronize data through a peer-to-peer network so job scheduler in each region has enough information to make a decision.

Job scheduler. A job scheduler plays the same role as Catalyst optimizer on Spark SQL [5]. It takes a data query and metadata of source datasets as input and generates a physical execution plan. Additionally, cost and network bandwidth are taken into account when selecting a plan in our design. In many cases, placement of jobs and intermediate results becomes a dominating factor instead of local I/O operations.

Job manager. A job manager persists the current execution plan generated by job scheduler and corresponding job states for each job. It propagates the execution plan and job states to all other job managers in remote regions. The job manager in each region fires one or more subtask executions according to the execution plan. After a subtask completes, the job manager invokes the job scheduler to re-evaluate the execution plan, and broadcasts job states and potential plan changes. The job manager may disrupt subtask execution due to a changed plan.

Task executor. A task executor consists of compute resources. It is built on top of existing cloud offerings to remove the resource constraints of some previous works [12]. Because there is no stable access pattern, and each job requires a different amount of resources, leveraging serverless functions (e.g., AWS Lambda[2]) as the main executor improves resource utilization [14]. Using spot VM instances to run the queries may save cost when incoming workloads can be well predicted.

Transient datastore. The results of subtasks are stored in a transient datastore. Transient data is not deleted right after it is passed to the next subtask as it may be used in later stages. The job manager scans the datastore regularly to identify transient data that is not referred by any execution plans after a certain amount of time and removes it to avoid unnecessary storage cost.

3 PRELIMINARY RESULTS

To verify the effectiveness of the proposed system, we have conducted an experiment on AWS. We generated a TPC-DS [16] dataset of 10GB. Among the dataset, We put 4 relatively small tables in an Amazon S3 [1] bucket of one of the regions, and we evenly distributed a large table to all 4 regions.

Table 1: Cost of Different Job Placement Strategies

Strategy	Compute($)	Network($)	Total($)
Aggregation	0.0396	0.2760	0.3155
In-place	0.1007	0.0893	0.1900
Optimized In-place	0.0720	0.0003	0.0723
Hybrid	**0.0580**	**0.0033**	**0.0614**

We simulated Query 7 of TPC-DS test suite in Python programs on EC2 c5d.2xlarge spot instances. We compared our hybrid strategy against 3 other strategies. **(a) Aggregation**: this strategy moves all data to the region with the lowest compute cost and does all computation there. **(b) Naive In-place**: we do as much computation as possible in each region without moving any data beforehand, and then we aggregate the intermediate results in one of the regions. **(c) Optimized In-place**: we add optimization to distribute the small tables to all regions before each region does its computation (the WAN-usage optimal solution). **(d) Hybrid**: We combined (a) and (c). Decisions are made based on prices of compute and data transfer. For example, for regions with high compute prices but relatively low network prices, original data should be transferred elsewhere to reduce cost.

Table 1 shows our hybrid strategy achieved an additional cost reduction by 15.1% versus optimized in-place computation. Although optimized in-place strategy is WAN-usage optimal, it does not consider the facts that the network prices between regions are heterogeneous and prices of compute vary in different regions. When price factors are taken into account, it becomes sub-optimal.

4 DISCUSSION

We identify the main challenges to build the proposed system.

Challenge#1 Design an efficient job scheduling algorithm. Existing query optimizers such as Catalyst [5] assign each operation a respective cost based on its amount of utilized resources, and search for the "best" execution plan. Execution of such an algorithm itself can be time-consuming and incur significant overheads. Taking actual cost and network bandwidth into account complicates the problem even further. An algorithm that quickly finds a good enough execution plan is needed.

Challenge#2 Estimate required resources for subtasks. Both spot VMs and serverless functions require a certain level of provisioning. Both over-provisioning and under-provisioning should be avoided if possible. Program analysis techniques can be applied to estimate minimum memory needed for a subtask to avoid exceeding allocated memory or allocating too much memory.

Challenge#3 Manage job states between regions. The cross-region data analytics engine must be fault-tolerant. All job states are shared among job managers. When the job manager in a region fails to function properly, other regions should be able to detect it and take over, which also means existing jobs must be migrated to other regions.

Challenge#4 Avoid unnecessary execution plan switching. There are clear benefits to adaptively switch to a new execution plan at runtime. However, if execution plan switching happens too frequently, a job may never complete.

REFERENCES

[1] Amazon Web Services 2021. Amazon S3. https://aws.amazon.com/s3/.

[2] Amazon Web Services 2021. AWS Lambda. https://aws.amazon.com/lambda/.

[3] Amazon Web Services 2021. Centralized Logging. https://aws.amazon.com/solutions/implementations/centralized-logging/.

[4] Amazon Web Services 2021. Regions and Availability Zones. https://aws.amazon.com/about-aws/global-infrastructure/regions_az/.

[5] Databricks 2021. Catalyst Optimizer. https://databricks.com/glossary/catalyst-optimizer.

[6] Tarek Elgamal. 2018. Costless: Optimizing cost of serverless computing through function fusion and placement. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 300–312.

[7] Anshul Gandhi and Justin Chan. 2015. Analyzing the Network for AWS Distributed Cloud Computing. *SIGMETRICS Perform. Eval. Rev.* 43, 3 (Nov. 2015), 12–15. https://doi.org/10.1145/2847220.2847224

[8] Kyungyong Lee and Myungjun Son. 2017. DeepSpotCloud: Leveraging Cross-Region GPU Spot Instances for Deep Learning. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. 98–105. https://doi.org/10.1109/CLOUD.2017.21

[9] Ming Mao and Marty Humphrey. 2011. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[10] Microsoft 2021. Azure geographies. https://azure.microsoft.com/en-us/global-infrastructure/geographies/.

[11] Netflix 2016. Completing the Netflix Cloud Migration. https://about.netflix.com/en/news/completing-the-netflix-cloud-migration.

[12] Suraj Pandey, Adam Barker, Kapil Kumar Gupta, and Rajkumar Buyya. 2010. Minimizing Execution Costs when Using Globally Distributed Cloud Services. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*. 222–229. https://doi.org/10.1109/AINA.2010.30

[13] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. 2015. Low Latency Geo-Distributed Data Analytics. *SIGCOMM Comput. Commun. Rev.* 45, 4 (Aug. 2015), 421–434. https://doi.org/10.1145/2829988.2787505

[14] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. 2019. Shuffling, Fast and Slow: Scalable Analytics on Serverless Infrastructure. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 193–206. https://www.usenix.org/conference/nsdi19/presentation/pu

[15] Snowflake 2021. Sharing Data Securely Across Regions and Cloud Platforms. https://docs.snowflake.com/en/user-guide/secure-data-sharing-across-regions-plaforms.html.

[16] TPC 2021. TPC-DS benchmark. http://www.tpc.org/tpcds/.

[17] Raajay Viswanathan, Ganesh Ananthanarayanan, and Aditya Akella. 2016. CLARINET: WAN-Aware Optimization for Analytics Queries. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 435–450. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/viswanathan