C2GEO
Techniques and Tools for Real-time Data-intensive Geoprocessing in Cloud Computing

Hassan A. Karimi and Duangduen Roongpiboonsopit
Geoinformatics Laboratory, School of Information Sciences, University of Pittsburgh
135 North Bellefield Avenue, Pittsburgh, U.S.A.

Keywords: Cloud computing, Geoprocessing, Real-time, Data-intensive, Geospatial data.

Abstract: Interest in implementing and deploying many existing and new applications on cloud platforms is continually growing. Of these, geospatial applications, whose operations are based on geospatial data and computation, are of particular interest because they typically involve very large geospatial data layers and specialized and complex computations. In general, problems in many geospatial applications, especially those with real-time response, are compute- and/or data-intensive, which is the reason why researchers often resort to high-performance computing platforms for efficient processing. However, compared to existing high-performance computing platforms, such as grids and supercomputers, cloud computing offers new and advanced features that can benefit geospatial problem solving and application implementation and deployment. In this paper, we present a distributed algorithm for geospatial data processing on clouds and discuss the results of our experimentation with an existing cloud platform to evaluate its performance for real-time geoprocessing.

1 INTRODUCTION

Cloud computing has received much attention in recent years. Enabled by tremendous advances in computing, storage, and networking, cloud computing has become one of the most promising developments towards the vision of utility computing. It offers applications an unprecedented, on-demand scalability. Much of the current research in cloud computing is focused on developing cloud infrastructures that can solve problems in a variety of disciplines. While elements of such infrastructures are in place, many application-specific issues in cloud computing, such as application/service optimization, still remain unresolved.

One class of applications is geospatial modeling, analysis, and simulation (geoprocessing). The coordination, collaboration, and sharing of geoinformation has been a long standing and challenging problem ever since Geospatial Information System (GIS) technology became available as a tool for geoprocessing. In the early years, the problem of managing such location-oriented information was less complex due to the limited amount of available geospatial data sources, geospatial data formats and structures. Today, the problem is extremely complicated and seemingly insurmountable due to the availability of numerous geospatial data sources, geospatial data in diverse and heterogeneous formats and structures, and various geospatial data collection technologies including geo-positioning and remote sensing. The problem is compounded by a paradigm shift from centralized geoprocessing, through stand-alone GIS software packages, to decentralized geoprocessing through Web services.

While the geospatial community has been active in addressing compute- and data-intensive geospatial problems by utilizing high-performance computing (HPC) platforms, primarily supercomputers and grids, over the past decade, there are many existing and emerging geospatial applications that have not yet been tackled through the HPC approach. One reason for this can be attributed to the fact that the utilization of existing HPC platforms often requires that domain scientists and engineers become computer experts by gaining complex computational knowledge and skills.

Using cloud computing for data-intensive, time-

A. Karimi H. and Roongpiboonsopit D..
C2GEO - Techniques and Tools for Real-time Data-intensive Geoprocessing in Cloud Computing.
DOI: 10.5220/0003394203710381
In *Proceedings of the 1st International Conference on Cloud Computing and Services Science* (CLOSER-2011), pages 371-381
ISBN: 978-989-8425-52-2

sensitive geospatial applications is advantageous over supercomputers and grids. Cloud computing provides much more suitable platforms for storing and manipulating very large map databases, typical in geospatial applications. In particular, clouds, unlike grids, are able to quickly scale up or down to the computing and storage needs of applications. This makes them an attractive alternative for deploying the next generation of data-intensive geoprocessing applications. An example application is location-based services (mostly available through smartphones) with a large number of current users, anticipated to increase by an order of magnitude in the next few years. However, for cloud computing to be useful in the geospatial community, there is a need for solutions and tools that specifically handle the unique characteristics of geoprocessing such as 3D indexing, retrieval, and computation techniques (uncommon in most current database management systems) and that are simple to utilize allowing scientists and engineers to focus on the problem at hand rather than trying to tweak and optimize complex codes.

In this paper, we focus on the class of geospatial applications that: (i) are increasingly becoming available on smartphones, (ii) involve very large databases (stored in RDBMSs), (iii) require data-intensive techniques for efficient geoprocessing, and (iv) require real-time response. The objective is to overcome the challenges of these applications using cloud computing. To that end, we present new techniques (distributed and parallel algorithms) suitable for real-time processing of data-intensive geospatial problems, called Cloud Computing for Geoprocessing (C2Geo). C2Geo is unique in several ways. First, despite the awareness and the availability of cloud computing, to date, geoprocessing in cloud computing has been limited to a handful of projects and there are no available techniques and tools for different classes of applications. Second, the real-time processing of data-intensive geospatial applications is expected to provide an insight into the capabilities and limitations of cloud computing paving the way to a better understanding of cloud computing as an emerging computing platform for problems across domains. C2Geo is intended to be scalable and provide high-performance geoprocessing automatically and transparently. C2Geo is expected to facilitate anywhere and anytime computing and provide means of solving compute- and data-intensive geoprocessing.

To emphasize the need for C2Geo, we evaluated a cloud computing platform for storing and retrieving large-scale Triangulated Irregular Networks (TINs) required for the real-time integrated Global Navigation Satellite System (iGNSS) QoS prediction for navigation services. Google App Engine (GAE) was chosen due to the fact that currently it is the only cloud computing platform available to researchers at no cost.

The contributions of the paper are development of a distributed algorithm for efficient processing of data-intensive geospatial applications in clouds, and evaluation of a cloud computing platform for real-time geospatial applications using navigation services as a case study.

The structure of the paper is as follows. Section 2 briefly overviews research studies and existing services/products that are geospatial related in clouds. Section 3 describes the concepts of C2Geo with a discussion of each module. Section 4 and 5 discusses and reported an evaluation of a cloud for handling large-scale TINs used in a real-time geospatial application. The paper ends with conclusions and future research in Section 6.

2 BACKGROUND

Geospatial applications typically require the access to and manipulation of large volumes of geospatial data. Recent trends show tremendous growth in geospatial data. This is due to two factors (Foerster et al., 2010): (i) the availability of more sophisticated data acquisition techniques and devices (e.g., airborne laser scanning, smartphones equipped with GPS, geo-enabled sensors) and (ii) advances in networking and Web technologies enabling unprecedented levels of data accessibility. In addition to being data intensive, geospatial applications often require intensive processing. Moreover, many of these applications inherently require that a large number of users (potentially anyone) be able to share data stored on several distant servers which translates into substantial data movement. The data, computing, and networking intensive nature of geospatial applications has made it challenging to achieve reasonable performance and scalability at affordable costs.

Until recently, most geospatial applications have been developed to run on desktops (Schäffer and Baranski, 2009). For applications that require higher levels of performance and scalability, two other options have been considered: supercomputers and grids. Supercomputers are often too costly for most geospatial applications. Grids usually achieve less than the required level of performance. With these

and other limitations of both supercomputing and grid computing, cloud computing is emerging as, potentially, the ideal paradigm for the development and deployment of geospatial applications. While still in its infancy, geospatial cloud computing is currently the focus of an intensive research and development efforts. Geospatial cloud computing was introduced as a specific type of cloud computing that focuses on adding geography to the cloud computing paradigm. It provides dynamically scalable geographic information technology, geospatial data, and geo-applications as a Web service/utility. It leverages the power of geography without needing the massive investment in specialized geo-data, people, and software typically required in an "enterprise GIS" implementation (Williams, 2009).

Cloud computing is now widely viewed as a promising paradigm for building tomorrow's geoprocessing systems (Brauner et al., 2009). Several projects both in academia and in industry have recently started efforts to develop prototypes of geospatial systems on clouds. For example, Cornillon (2009) explored the suitability of cloud computing for processing large volumes of satellite-derived sea surface temperature data. Hill (2009) presented the results of experiments using Amazon's Elastic Compute Cloud (EC2) for ocean-atmosphere modeling. Blower (2010) presented an implementation of a Web map service for raster imagery using the GAE environment. Wang et al. (2009) describe a prototype for retrieving and indexing geospatial data developed for GAE. In parallel with these efforts in academia, several vendors of GIS software have recognized the promise of cloud computing and some have already introduced cloud-based GISs. ESRI currently provides preconfigured ArcGIS Server Machine Images (AMI) for use in the Amazon Cloud infrastructure (ESRI, 2010). Running ArcGIS Server on Amazon allows organizations to deploy ArcGIS Server across more than one data center and access to Amazon's elastic computing infrastructure. In addition, ESRI's MapIt features Spatial Data Assistant (SDA) and Spatial Data Service (SDS) for Microsoft Windows Azure as in its current version it is unable to manage and process geospatial data (ESRI, 2009). Another example is Omnisdata's GIS Cloud (Omnisdata, 2010). GIS Cloud is a Web-based GIS powered by cloud computing with advanced capability of creating, editing, uploading, sharing, publishing, processing and analyzing geospatial and attribute data. Kim and MacKenzie (2009) used Amazon's EC2 in a climate change study with the purpose of calculating the number of days with rain in a given month on a global scale over the next 100 years. The computation used 70 gigabytes of daily sets of climate projection data. It took about 32 hours to process 17 billion records.

Cloud computing is seen as the needed paradigm to finally shift the (often intensive) processing part of geospatial applications from the desktop to distributed spatial data infrastructures (SDIs) (Schäffer and Baranski, 2009). By outsourcing computing and/or data intensive tasks to the cloud, geospatial applications will benefit in terms of performance, scalability, and startup cost.

While most initial research has concluded that cloud computing is a viable paradigm for compute- and data-intensive geoprocessing, the fundamental limitation remains that cloud infrastructures are, in general, developed for generic computing; they often are not aware of the spatial nature of the data. As a result, existing cloud computing infrastructures still require extensive research to develop optimization techniques that would lead to true geospatial clouds.

3 C2GEO TECHNIQUES & TOOLS

C2Geo is a set of techniques and tools designed for the efficient processing of real-time data-intensive geospatial applications. Considering the continual demand for geospatial applications, cloud computing providers can implement C2Geo in their clouds, as part of geospatial database handling and geoprocessing, in order to meet the requirements of real-time data-intensive geospatial applications, especially those that involve a large number of users with mobile devices (e.g., smartphones).

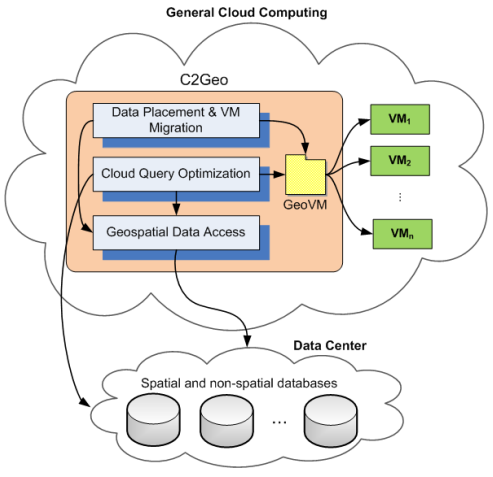

Figure 1: C2Geo in a cloud.

Figure 1 shows the concept of C2Geo in clouds. C2Geo encompasses three main modules: data placement and Virtual Machine (VM) migration, optimal query processing, and geospatial data access. A VM in the context of geoprocessing contains geospatial data and operations on the data that can be processed on any physical machine. A master VM, called Geospatial Virtual Machine (GeoVM), acts as a directory of all VMs (geospatial data and processes).

The objective of the data placement and VM migration module is to place geospatial data in the most suitable way for the usage pattern of real-time data-intensive geospatial applications. The objective of the optimal query processing module, Cloud Query Processing (CQO), is to discover VMs through GeoVM. The objective of the geospatial data access module is to provide a suitable indexing strategy for optimal retrieval of geospatial data based on locations of VMs.

3.1 Data Placement and VM Migration

Two key factors determine the performance of a data-intensive application in a cloud: data placement and VM deployment and migration. Data placement determines the location of the data being accessed. VM deployment and migration determine where the VMs must be initially deployed and, if necessary, when and where they must be subsequently relocated (Sato et al., 2009). Although they generally contribute to the same purposes (higher performance, improved availability, and better fault tolerance), data placement and VM migration are considered, in most current research, as two independent mechanisms. As a result, most existing data placement and VM relocation solutions are sub-optimal. For this, algorithms that significantly reduce response time through a novel approach where data placement and VM relocation are accomplished in tandem are needed. Specifically, these algorithms should simultaneously take into account several criteria relevant to the geospatial nature of data and to the workload including: (i) location of users, (ii) correlation between data and users' queries, (iii) load distribution on servers, (iv) network parameters (e.g., bandwidth of links, congestion), and (v) mobility of users. We argue that by simultaneously considering these and other relevant factors, it will be possible to achieve far higher performance, availability, and fault tolerance.

Static data placement solutions are not able to efficiently adapt to dynamic changes in the cloud, e.g., increase in the workload at some servers and congestion in some areas of the network. The task of placement is further complicated by the issues of shared data, data inter-dependencies, application changes and user mobility (Agarwal et al., 2010). Because of the limitations of static data replication, a few recent research efforts have introduced dynamic replication schemes. These include Skute (Bonvin et al., 2009), Re:FRESHiT (Voicu et al., 2010), and Volley (Agarwal et al., 2010). In Skute, the number of replicas is dynamically adapted to the query load while maintaining availability guarantees in case of failures. In Re:FRESHiT, the focus is on managing replicas of frequently updated data. Because of the high cost of updating all replicas, the proposed protocol, i.e., Re:FRESHiT, organizes read-only sites in virtual trees based on the sites' freshness levels, and introduces a routing mechanism for reading data, while at the same time allowing users to specify their own freshness requirements. Trees are automatically reorganized after individual nodes are refreshed or when new replicas join.

3.2 Cloud Query Optimization

CQO is based on the assumptions that the geospatial data is stored in RDBMSs and the data is available through VMs. To find an optimal query processing in C2Geo, CQO first, through GeoVM, discovers all relevant VMs, i.e., locations of geospatial data components and the processes on them. Then, using the requirements of the query and the locations of the required VMs, it finds an optimal geoprocessing strategy, which will then be passed on to the cloud. One goal of CQO is to minimize response time and power consumption. A second goal of CQO is to minimize computing load on smartphones, which means pushing geoprocessing to the cloud as much as possible. In our previous work, we experimented with some of the techniques in CQO using grid platforms (see Liu and Karimi, 2008).

CQO consists of two main modules, resource selection and parallelism processing, and three auxiliary processes. Optimizing queries for clouds is challenging as it entails a large search space that decreases the overall performance. To overcome this problem, CQO limits the search space by selecting a subset of VMs in the resource selection module thus improving optimization time performance. The resource selection module is based on a ranking function that incorporates several performance-related factors. Available VMs are ranked by their costs for a specific operation and the one with the least cost is selected for executing the operation. The resource selection module helps CQO reduce

optimization cost without excluding potential superior computing resources. To further improve query response time, CQO exploits parallelism by detecting data dependency among operations in the parallelism processing module. Before discussing the two main modules of CQO, auxiliary services that supply statistics and other types of information for query optimization are presented next.

3.2.1 Input and Auxiliary Services

The input to CQO is represented in a tree structure called the Abstract Query Tree (AQT). Each leaf node in an AQT is an equi-join with two base relations and an internal node is an operation on the results of its leaf nodes. There are three auxiliary services built in CQO to provide run-time information for query optimization: Environment Information Service (EIS), Database Information Service (DIS), and Transmission Prediction Service (TPS). EIS is responsible for providing both static and dynamic information about a given VM, such as system workload in a percentage rate, CPU speed, and RAM amount. DIS manages a catalog of existing replicas of base relations in the cloud and retrieves them back to a client upon request. For a given relation, DIS can provide VMs that maintain a replica of the relation and statistics about the relation such as relation size and field size. For a relation in a given query, TPS is responsible for estimating a candidate VM's transmission performance with respect to other VMs involved in that query. Mean transmission latencies between VMs from historical data are often used to measure a VM's transmission capacity. But a problem with this approach is that mean values can be significantly affected by data distribution—outliers with arbitrarily high or low values can greatly impact mean values. In CQO, an index, Transmission Latency Reputation (TLR), is designed to reduce such inaccuracy. The calculation of TLR is as follows.

Suppose that the query Q to be executed during a time period t involves relations R_1, R_2, …, R_i, ..., R_N. Relation R_i has M_i replicas that are located at VMs H_{i1}, H_{i2}, …, H_{ij}, …, H_{iMi}, respectively. The TLR of VM H_{ij} for Relation R_i in query Q during t is computed as a weighted mean:

$$TLR_{ij} = \frac{\sum_{k=1}^{N}\sum_{l=1}^{M_k} w_{(ij,kl)} * TL_{(ij,kl)}}{\sum_{k=1}^{N}\sum_{l=1}^{M_k} w_{(ij,kl)}} \ (k \neq i) \quad (1)$$

where $TL_{(ij,kl)}$ is the mean transmission latency between H_{ij} and H_{kl} during t. If, for H_{kl} and H_{ij}, relation R_i is the only relation they have that is involved in Q, H_{kl} should not be taken in computing TLR_{ij}. The reason for excluding such VMs from the calculation is that there will be no transmission between that VM and the VM used in the computation while executing the query; they compete to be the provider of R_i. $w_{(ij,kl)}$ is the weight assigned to H_{ij} by comparing $s^2_{(ij,kl)}$, the variance of transmission latencies between H_{ij} and H_{kl}, with the maximum of variances of transmission latencies between H_{ij} and other VMs, noted as $max_{ij}(s^2)$:

$$w_{(ij,kl)} = \frac{s^2_{(ij,kl)}}{\max_{ij}(s^2)} \quad (2)$$

Plugging Equation (2) into (1) simplifies the latter to:

$$TLR_{ij} = \frac{\sum_{k=1}^{N}\sum_{l=1}^{M_k} s^2_{(ij,kl)} * TL_{(ij,kl)}}{\sum_{k=1}^{N}\sum_{l=1}^{M_k} s^2_{(ij,kl)}} \ (k \neq i) \quad (3)$$

By introducing $w_{(ij,kl)}$, the mean transmission latency of a VM is adjusted; the more outliers a VM has in its latency data and the larger values of outliers, the larger will be the value of TLR and thus the smaller its transmission capacity with respect to other VMs. The ranking function discussed in the following section uses TLR as an index of a VM's transmission capacity when selecting resource providers.

3.2.2 Resource Selection

In the resource selection module, an AQT is recursively visited in order to find candidate VMs for each base relation in the AQT. This process is performed in a sub-module called resource locator. For each base relation, the resource locator contacts DIS which will return a list of candidate VMs containing the requested base relation. The returned list is passed to a sub-module called resource ranker. The resource ranker checks with DIS, TPS, and EIS, respectively, to obtain both static and dynamic statistics about all candidate VMs. A ranking function based on the cost function proposed by Mackert and Lohman (Mackert and Lohman, 1986) is used in the resource ranker to compute the rank of a given candidate VM H_{ij} for a base relation R_i. The ranking function is a linear combination of weighted and standardized values of five factors:

$$rank_{ij} = \frac{(mips_{ij} - \min(mips)) \times w_{cpu}}{\max(mips) - \min(mips)} + \frac{(ram_{ij} - \min(ram)) \times w_{ram}}{\max(ram) - \min(ram)} + \frac{(count_{ij} - \min(count)) \times w_{count}}{\max(count) - \min(count)} + \frac{w_{wk}}{(wk_{ij} - \min(wk))/(\max(wk) - \min(wk)) + 1} + \frac{w_{TLR}}{(TLR_{ij} - \min(TLR))/(\max(TLR) - \min(TLR)) + 1} \quad (4)$$

where $mips_{ij}$ is MIPS (Million Instructions Per Second) of H_{ij}; w_{mips} is weight of MIPS; ram_{ij} is RAM amount at H_{ij} (MB); w_{ram} is weight of RAM; $count_{ij}$ is number of relations that are involved in a query and maintained by H_{ij}; w_{count} is weight of count; wk_{ij} is current workload of H_{ij} (0 means idle and 1 means that H_{ij} is fully utilized); w_{wk} is weight of workload.

The introduction of *count* into the ranking function increases the chances of VMs with multiple relations getting higher ranks. The higher rank a VM receives, the more likely it performs joins locally. For special running environments where certain factors become dominant or subordinate, corresponding weights can be adjusted accordingly.

After computing the ranks of all candidate VMs for a base relation, the VM with the highest rank is chosen as the provider of the base relation and added to the AQT. Since the factors in the ranking function are those that will greatly impact the performance of a query execution, it can be argued that the rank a VM receives reflects the fitness of a VM as a candidate for a base relation in a given query. The higher the rank, the higher are the chances of a VM to be chosen as the provider of the relation. Once all relations are allocated to specific VMs, the AQT will be transformed into a Physical Query Tree (PQT) which in turn is passed to the parallelism processing module for further optimization.

3.2.3 Parallelism Processing

We have developed a strategy for parallelism processing which alleviates the burden of Message Passing Interface (MPI) development on geospatial developers. Before checking parallelism in a PQT, information about intermediate results (i.e., result of a join that is part of the input to another join) needs to be known since CQO needs to determine data transfers between operations. This information includes: number of records, record size, number of blocks, and number of distinct values and index height of the join field. These statistics are temporarily added on to the information repositories in DIS and are removed once an execution plan for a query is determined.

After the resource providers are selected, CQO checks to see if it is possible to exploit parallelism during the execution of the query. Based on the dependency among the operations of a query, parallel processing in clouds can be achieved in different forms: single operation single data (SOSD), single operation replicated data (SORD), multiple operations single data (MOSD), and multiple operations multiple data (MOMD).

Implementing SOSD or MOSD involves replicating a data set in one or more VMs. With large volumes of data in clouds (terabytes) and VMs connected via local and wide area networks, data replication could introduce an overhead that may reduce parallelism gains. Thus, SOSD and MOSD are not considered in CQO. As for SORD, it requires knowledge of domain-specific operations (e.g., how to partition a data set and how to combine results from parallel processes). For instance, in building a parallel routing algorithm to obtain a best path, Karimi and Hwang (1997) suggest to partition a network into equal sub-networks in order to maximize load balancing and improve performance. Such an algorithm can be implemented in an independent module and plugged into CQO.

```
Input: PQT
Output: an ordered set of sequential steps
Variables:
      Set N; Set S;
      Node n;
Begin
      Name intermediate results;
      Estimate statistics of intermediate results (using DB information
service);
      While there is a node in PQT do
      Begin
            Create new sequential step s;
            N ← all leaf nodes;
            For i = 1 to #of nodes in N do
            Begin
          n ← N[i];
          If n does not have dependency of any other node in N then
                Add the operation at n into s;
          Remove n from N;
          Remove n from PQT;
            End;
            Add s to S;
      End;
      Return an ordered set of sequential steps;
End.
```

Figure 2: Algorithm to detect MOMD parallelism.

The algorithm in Figure 2 is proposed to implement MOMD. Since MOMD exists when there is no data dependency between operations, it can only be found between operations in the leaf nodes of a PQT. The algorithm checks all operations in all leaf nodes in a PQT for data dependency. Operations with data independency are removed from the PQT and are added to a new sequential step. This process is repeated until all joins in the PQT are processed.

The outcome is an ordered set of sequential steps.

After parallel executions are determined for each step, a structure called "parallelism-inside-of-sequential-steps" (PSS) is used to represent the output of optimization (i.e., an execution plan). In PSS, an execution plan is formed by a series of sequential steps. Each sequential step includes a set of operations that is scheduled to run in parallel and hence the name is "parallelism-inside-of-sequential-steps". Operations in one sequential step have to wait for the operations in the previous steps to be completed so that all their input data become available. If no parallelism is possible in a query, for instance, a left-deep join, a sequential step only has one operation (e.g., a join). Upon determination of the execution order of operations in PSS and allocation of each operation with specific resources, PSS is ready to be submitted for execution.

3.3 Geospatial Data Access

In this module, we are developing novel techniques for geospatial data indexing and retrieval in cloud environments. Specifically, we focus on four major directions: (i) geospatial indexing, (ii) location-aware data placement, (ii) network-aware indexing, and (iv) access-based data reindexing, replication, and migration.

3.3.1 Geospatial Indexing

Geospatial indexing has been an active research area for many years. In particular, several spatial indexing techniques have been developed for distributed systems. Substantial efforts focused on adapting well studied spatial indexing techniques such as kd-Trees (Robinson, 1981), Quadtrees (Finkel and Bentley, 1974, Samet, 1984), R-Trees (Guttman, 1984), and Octrees (Hunter, 1978, Reddy and Rubin, 1978) to the context of distributed systems. For example, Zimmermann et al. (2004) presented an architecture to efficiently route and execute spatial queries based on distributed R-tree and Quadtree structures. The architecture supports both spatial range and *k* nearest neighbor queries.

With more data-intensive applications being hosted in clouds, research has recently shifted to developing spatial indexing techniques specifically designed for clouds (e.g., see Mouza et al. 2007, 2009; Wang et al., 2010; Wu and Wu, 2009. Mouza et al. (2007, 2009) propose a new indexing structure, called SD-Tree (Scalable Distributed Rtree), with the objective of evenly balancing the utilization of the processing power and storage of a pool of distributed data servers. Wu and Wu (2009) presented an indexing framework for clouds where processing nodes are organized in a structured overlay network, and each processing node builds its local index to speed up data access. A global index is built by selecting and publishing a portion of the local index in the overlay network. The global index is distributed over the network, and each node is responsible for maintaining a subset of the global index. Wang et al. (2010) integrate a CAN-based routing protocol (Ratnasamy et al., 2001), called RT-CAN, and an R-tree based indexing scheme to support efficient multi-dimensional query processing in a cloud system. RT-CAN organizes storage and compute nodes into an overlay structure based on an extended CAN protocol and supports the processing of multi-dimensional range and KNN queries.

A fundamental issue is how to store the spatial index on the cloud. For example, assume that a Quadtree index is used. Data is first partitioned until it becomes possible to store one or more quadrant(s) on a server of the cloud. A simple alternative is to store the entire index on a single node. Another alternative offering improved scalability and availability could be to distribute the index on an overlay formed by some or all of the cloud's nodes.

3.3.2 Location-Aware Data Placement

Efficient data placement aims at two objectives: (i) reducing disk I/O cost needed for data retrieval and (ii) reducing communication cost associated with the retrieval operation. To illustrate, assume that Quadtree indexing is used. A simple approach to reduce disk I/Os is to partition data so that the maximum number of the most frequently accessed quadrants can be stored in the main memory of the cloud's servers. Reducing communication cost requires that the data be stored where it is most often accessed. A natural way of achieving this in geospatial applications is to store geospatial data on servers located in the area referenced by the data itself. The intuition is that, in most geospatial (location-based) applications, users access geospatial data relevant to their current location.

A possible approach for data placement is location-aware data placement (LDP). Intuitively, LDP consists of partitioning and distributing geospatial data on cloud's nodes so that every data unit is stored on a server that is as close as possible to the area referenced by that data unit. On a given server, the memory constraints are also considered when determining the size of data units (e.g., quadrants).

3.3.3 Network-Aware Spatial Indexing

It is well known that, in a distributed environment, the search complexity is dominated by the communication overhead between servers rather than by I/O operations at individual servers (Zimmermann et al., 2004). To achieve near real-time latency, it is therefore crucial to take into account the network characteristics when deploying the distributed spatial index on the cloud's nodes. We introduce the concept of *network-aware spatial indexing*. In this new indexing paradigm, the index's partitions are stored on the nodes such that the average *indexing latency*, time to reach the relevant key in the index tree structure, is minimized.

3.3.4 Access-based Data Reindexing, Replication and Migration

In cloud computing, a substantial share in query processing time is the time to access the indexes and the time to retrieve the data. In location-based applications, performance may deteriorate significantly if a *static* index and data distribution/replication scheme is adopted. In addition to *location*, other temporal parameters can also have an impact on response time in many geospatial applications. Examples include: the time of the day (day vs. evening), day of the week (working day, week-end day), public events, weather, holidays, period in the year, etc. These spatial and temporal parameters directly determine the query load submitted to an application. For example, on a hot week-end summer day, a significantly higher than usual number of queries are likely to be submitted to a navigation application by users driving on a highway on their way to the beach. Because of the dynamic nature of queries, a static distribution/replication of data and indexes is likely to yield sub-optimal performance.

4 CLOUD EVALUATION

Since implementation of C2Geo on each cloud platform requires an understanding of some of its techniques and the availability of certain tools to utilize the cloud effectively, a comprehensive evaluation of C2Geo is beyond the scope of this paper. However, to illustrate some of the potential issues that may arise while implementing C2Geo on clouds, we evaluated the performance of an existing cloud computing for real-time geospatial applications.

We have chosen navigation applications, one type of location-based services, as a representative real-time data-intensive geoprocessing. Navigation applications have a usage pattern that is ideal for cloud computing as it can adequately scale (down or up) to multiple simultaneous users (very small to very large numbers) with performance appropriate for the real-time response. The application we are focusing on is a real-time prediction module for quality of services for iGNSS (iGNSS QoS) with real-time processing constraints. iGNSS QoS prediction requires large-scale TINs for satellite visibility calculation, which is a real-time process. Thus, there is a need for an efficient strategy to retrieve large-scale TINs from a cloud.

GAE was chosen due to its publicly available service at free (or low) cost and its full featured platform that allows developers to test their web applications on a cloud platform in a short time. However, the current GAE does not natively support geospatial data and processing. An open-source project, called GeoModel, for GAE was used to index geospatial data and perform basic spatial operations (i.e., proximity and bounding box) (Nurik and Shen, 2009).

A TIN, covering the University of Pittsburgh's main campus and the surrounding neighbourhoods with a 3.048 km by 3.048 km area, was created from LiDAR point cloud. The LiDAR point cloud has a point spacing of 1 m and the total number of LiDAR points is about 3.4 million. Since the GAE datastore is a schemaless or non-relational database, the created vertices and triangles were uploaded to GAE database as vertex (or point) and triangle entities. Point entities were uploaded through the use of GeoModel, which it defines a geocells property for spatial indexing, while triangles were uploaded as generic entities with no geocell attached.

Each point and triangle entity has a unique key assigned by the GAE for expediting the search. Point entity also has a property that contains a list of triangle IDs that have the point as their vertex. Each triangle entity has a property that contains a list of point IDs used as its vertices. To retrieve a TIN from the GAE datastore for a querying area, we used a two-step approach: (1) retrieve point entities using geocells generated by GeoModel and (2) retrieve triangles associated with the retrieved point entities using triangle IDs.

Due to GAE's limited quota on the total storage space (1 GB), only a small part of the prepared TIN could be stored in the GAE datastore, which covers 100 m x 3,048 m containing 225,369 vertices and 226,120 triangles.

The performances of GAE with the use of GeoModel were measured for performing proximity and bounding box queries. Various sizes and locations of the two query types were used in this evaluation, which are reasonable sizes for visibility calculation of iGNSS QoS prediction. For proximity, queries with the proximity distance of 10, 20, 30, and 40 m at 10 different locations (within the boundary of the uploaded) were created. For bounding box, long narrow strips with the size of the 100, 200, 300, 400, and 500 m length by the 1-m width were created at 5 different locations.

5 RESULTS AND DISCUSSION

For proximity search, only 32 of 40 defined queries could be completed due to the 30-second request limit imposed by GAE. A majority of elapsed times were caused by GeoModel. GeoModel determined relevant geocells, retrieved point entities of the computed geocells through the GAE datastore, and calculated and sorted the retrieved entities by distance, then returned a querying result. Searching entities from a string of geocells, which is an attribute not a key of entities, is not an optimal approach provided by GAE. In addition, the internal process of GeoModel of calculating and sorting distance is sequential for each query thread. Thus, increasing the proximity distance tends to decrease the performance. In addition, high variation of elapsed times can be observed due to the nature of sharing resources in the cloud. The circle symbols in Figure 3 show the elapsed time by the first-step of TIN query (point entities) for proximity queries. The second-step of TIN query (triangle entities) required relatively short time, about 1,3,5,9 seconds for the proximity distance of 10 to 40 m, respectively.

For bounding box search, all the defined sizes were completed within the 30-second limit. The square symbols in Figure 3 show the elapsed time by the first-step of TIN query (point entities) for bounding box queries. Again, the second-step of TIN query (triangle entities) required relatively short time, about 1,2,3,4, and 5 seconds for the bounding boxes with length of 100 to 500 m, respectively. The performances varied greatly according to the sizes of the bounding boxes. The long latency was mainly caused by the process of retrieving points for the computed geocells that contain the bounding box.

In summary, GAE is a general-purpose cloud computing platform that even though it provides a full set of features for easily developing web applications, it does not natively support storing, indexing, and retrieving geospatial data. This makes geoprocessing on an existing cloud, like GAE, more challenging and requiring more efforts than other Web applications with generic types of data. Therefore, techniques and tools like C2Geo are expected to facilitate development of geospatial applications in cloud and enhance the performances of real-time geoprocessing.

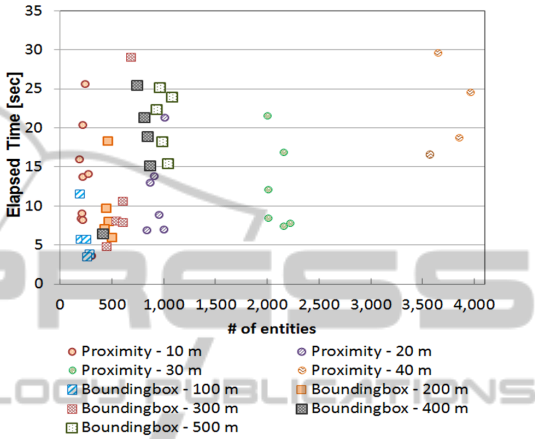

Figure 3: Evaluation performance of GAE for retrieving TIN data.

6 CONCLUSIONS & FUTURE RESEARCH

This paper discusses the result of evaluation of the Google App Engine cloud for addressing the requirements of iGNSS QoS prediction, a real-time geospatial application.

A large-scale TIN was used for testing the geospatial data retrieval performance of GAE with GeoModel. The result reveals that the current GAE platform and available tools are not ready yet to handle efficiently some of the data- and/or compute-intensive problems in real-time geospatial applications. There are several reasons for this. One is that the current GAE platform offers limited techniques and tools for geoprocessing.

Research in utilizing cloud computing for real-time geoprocessing should address the following: development of geoprocessing techniques and tools specifically designed for cloud implementation and deployment, such as C2Geo; development of tools that allow developers flexibility in using cloud resources for geospatial applications.

REFERENCES

Agarwal, S., Dunagan, J., Jain, N., Saroiu, S., Wolman, A. and Bhogan, H. (2010). Volley: Automated data placement for geo-distributed cloud services. In *7th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA.

Blower, J. (2010). GIS in the cloud: implementing a Web Map Service on Google App Engine. In *1st Intl. Conf. on Computing for Geospatial Research & Applications*, Washington D.C.

Bonvin, N., Papaioannou, T. and Aberer, K. (2009). Dynamic cost-efficient replication in data clouds. In *1st workshop on Automated control for datacenters and clouds*, Barcelona, Spain, 49-56.

Brauner, J., Foerster, T., Schaeffer, B. and Baranski, B. (2009). Towards a research agenda for geoprocessing services. In *12th AGILE International Conference on Geographic Information Science*, Hanover, Germany.

Cornillon, P. (2009). Processing large volumes of satellite-derived sea surface temperature data - is cloud computing the way to go? In *Cloud Computing and Collaborative Technologies in the Geosciences Workshop*, Indianapolis, IN.

ESRI, (2009). Spatial data service deployment utility for Windows Azure is available! Retrieved from: http://blogs.esri.com/Dev/blogs/mapit/archive/2009/12/18/Spatial-Data-Service-Deployment-Utility-for-Windows-Azure-is-available_2100.aspx.

ESRI, (2010). ArcGIS and the cloud Retrieved from: http://www.esri.com/technology-topics/cloud-gis/arcgis-and-the-cloud.html.

Finkel, R., Bentley, J. (1974). Quad trees a data structure for retrieval on composite keys. *Acta informatica,* 4(1), 1-9.

Foerster, T., Schaeffer, B., Baranski, B. and Lange, K. (2010). Geoprocessing in hybrid clouds. In *Geoinformatik*, Kiel, Germany.

Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, Boston, Massachusetts, 47-57.

Hill, C. (2009). Experiences with atmosphere and ocean models on EC2. In *Cloud Computing and Collaborative Technologies in the Geosciences Workshop*, Indianapolis, IN.

Hunter, G. (1978). Efficient computation and data structures for graphics. Princeton University, Princeton, NJ, USA.

Karimi, H. A., Hwang, D. (1997). A Parallel Algorithm for Routing: Best Solutions at Low Computational Costs. *Geomatica,* 51(1), 45-51.

Kim, K. S., MacKenzie, D. (2009). Use of cloud computing in impact assessment of climate change. In *Free and Open Source Software for Geospatial (FOSS4GT)*, Sydney, Australia.

Liu, S., Karimi, H. (2008). Grid query optimizer to improve query processing in grids. *Future Generation Computer Systems,* 24(5), 342-353.

Mackert, L. F., Lohman, G. M. (1986). R* Optimizer Validation and Performance Evaluation for Distributed Queries. In *the Twelfth International Conference on Very Large Data Bases*, Kyoto.

Mouza, C. D., Litwin, W. and Rigaux, P. (2007). SD-Rtree: A scalable distributed Rtree. In *IEEE 23rd International Conference on Data Engineering (ICDE)*, Istanbul, Turkey, 296-305.

Mouza, C. D., Litwin, W. and Rigaux, P. (2009). Large-scale indexing of spatial data in distributed repositories: the SD-Rtree. *The VLDB Journal,* 18(4), 933-958.

Nurik, R., Shen, S., (2009). Geospatial Queries with Google App Engine using GeoModel Retrieved from: http://code.google.com/apis/maps/articles/geospatial.html#geomodel.

Omnisdata, (2010). GIS Cloud beta: the next generation of GIS Retrieved from: http://www.giscloud.com/.

Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Schenker, S. (2001). A scalable content-addressable network. In *ACM SIGCOMM Computer Communication Review*, San Diego, CA, USA, 161-172.

Reddy, D., Rubin, S., 1978. Representation of three-dimensional objects (No. CMU-CS-78-113). Pittsburgh, PA: Computer Science Department, Carnegie-Mellon University.

Robinson, J. (1981). The KDB-tree: a search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, Ann Arbor, Michigan, 10-18.

Samet, H. (1984). The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR),* 16(2), 187-260.

Sato, K., Sato, H. and Matsuoka, S. (2009). A model-based algorithm for optimizing I/O intensive applications in clouds using VM-based migration. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, Shanghai, China, 466-471.

Schäffer, B., Baranski, B. (2009). Towards spatial related business processes in SDIs. In *12th AGILE International Conference on Geographic Information Science*, Hannover, Germany.

Voicu, L. C., Schuldt, H., Breitbart, Y. and Schek, H.-J. (2010). Data and flexible data access in a cloud based on freshness requirements. In *3rd IEEE International Conference on Cloud Computing (CLOUD2010)*, Miami, FL, USA, 45-48.

Wang, J., Wu, S., Gao, H., Li, J. and Ooi, B. C. (2010). Indexing multi-dimensional data in a cloud system. In *ACM SIGMOD/PODS Conference*, Indianapolis, IN, USA.

Wang, Y., Wang, S. and Zhou, D. (2009). Retrieving and indexing spatial data in the cloud computing environment. *Lecture Notes in Computer Science, Cloud Computing,* 322-331.

Williams, H. (2009). A new paradigm for geographic information services. *Spatial Cloud Computing (SC2), White Paper*.

Wu, S., Wu, K.-L. (2009). An indexing framework for efficient retrieval on the cloud. *IEEE Data Engineering,* 32(1), 75-82.

Zimmermann, R., Ku, W. and Chu, W. (2004). Efficient query routing in distributed spatial databases. In *12th annual ACM international workshop on Geographic information systems*, Washington DC, USA, 176-183.