Parallel SECONDO: A Practical System for Large-Scale Processing of Moving Objects

Jiamin Lu
Faculty of Mathematics and Computer Science,
FernUniversität Hagen
Hagen, Germany
jiamin.lu@fernuni-hagen.de

Ralf Hartmut Güting
Faculty of Mathematics and Computer Science,
FernUniversität Hagen
Hagen, Germany
rhg@fernuni-hagen.de

***Abstract*—Parallel SECONDO scales up the capability of processing extensible data models in SECONDO. It combines Hadoop with a set of SECONDO databases, providing almost all existing SECONDO data types and operators. Therefore it is possible for the user to convert large-scale sequential queries to parallel queries without learning the Map/Reduce programming details.**

This paper demonstrates such a procedure. It imports the data from the project OpenStreetMap into SECONDO databases to build up the urban traffic network, and then processes network-based queries like map-matching and symbolic trajectory pattern matching. All involved queries were stated as sequential expressions and time-consuming in single-computer SECONDO. However, they can achieve an impressive performance in Parallel SECONDO after being converted to the corresponding parallel queries, even on a small cluster consisting of six low-end computers.

I. INTRODUCTION

In the past decade, along with the popularization of portable positioning devices like navigators and smart phones, a large amount of trajectory data (moving objects) are collected and various database technologies are proposed to analyze them for different purposes. SECONDO [7, 8] was developed under this background. It is designed as an extensible database system, providing a large number of data types [6] and algorithms [9] to represent and process moving objects efficiently.

Nowadays, like the many other databases, SECONDO is facing the challenges from the big data, since it was designed as a single-computer system and its capability is restricted by the underlying computer resources. Regarding the issue of large-scale data processing, parallel frameworks like MapReduce [5] and its open-source implementation Hadoop [1] are proposed in the recent years, in order to help the user to analyze massive amounts of data on a large cluster composed by cheap and low-end computers. However, such platforms usually lay more stress on keeping the balanced workload and fault-tolerance in heterogeneous systems. Instead, their programming paradigms are low-level and rigid, making custom user code difficult to maintain and reuse. Besides, they also lack the capability of processing special data types like moving objects.

For all these reasons and also inspired by the work like HadoopDB [4], Parallel SECONDO [10] is proposed. It is built up as a hybrid parallel system by combining Hadoop and a set of SECONDO databases. On one hand, it scales up the capability of processing special type data in SECONDO on a cluster of computers, so as to improve the efficiency of processing large-scale queries. On the other hand, it keeps the front-end and the executable language in SECONDO to make the end-users handle Parallel SECONDO still like a single-computer system.

In the rest of this paper, we first introduce the infrastructure and the data model of Parallel SECONDO in the second section. Afterwards, a practical example is presented in the third section to demonstrate the procedure of using Parallel SECONDO on solving realistic problems.

II. PARALLEL SECONDO

Parallel SECONDO uses the Hadoop framework to apply and schedule tasks to a set of SECONDO databases that are distributed on the cluster. Each task encapsulates a concrete query which will be fully processed by SECONDO in order to achieve the best performance.

The basic processing unit in Parallel SECONDO is called Data Server (DS), consisting of a compact SECONDO system named Mini-SECONDO and its database storage. Since nowadays it is common that a commodity computer also provides powerful computing resources, like multiple-core processors, several hard disks and a large amount of memory, the user can set several DSs on the same computer in order to fully use the system. On every cluster node, one DS is denoted as the Main Server (MS) for setting up the Hadoop node. The Hadoop Distributed File System (HDFS) is only used as the communication layer to schedule tasks and control the workflow of parallel queries. In addition, a simple distributed file system PSFS (Parallel SECONDO File System) is developed to exchange intermediate data among DSs directly, in order to improve the data transportation performance.

In Parallel SECONDO, one computer is set to be the master node of the Hadoop framework and its MS is set to be the *master Data Server* (mDS) of the whole system. The Mini-SECONDO within the mDS is called the *master database*. All the other DSs are set as *slave Data Servers* (sDSs), and their Mini-SECONDO systems are called *slave databases*.

The existing SECONDO text and graphical interfaces are kept unchanged in Parallel SECONDO. With them, the user can access the system by simply connecting to the *master database*. Besides parallel queries, the *master database* processes the common sequential queries as usual. Therefore, Parallel SECONDO keeps the underlying cluster, the Hadoop framework and all *slave databases* invisible to the end-user, so as to integrate Parallel SECONDO with the conventional single-computer SECONDO system seamlessly.

A. Auxiliary Tools

Parallel SECONDO also proposes a set of auxiliary tools to help the user to easily install and manage the system on large-scale clusters.

For example, the *ps-cluster-format* helps the user to install Parallel SECONDO on the cluster, including the Hadoop framework and all DSs. Besides, the *ps-secondo-buildMini* is provided to distribute Mini-SECONDO to DSs immediately, in case there is any new feature extended in the single-computer SECONDO system. Further, the *ps-startTTYCS* enables the user to access any Mini-SECONDO database within DSs. Besides the tools that help the user to deploy Parallel SECONDO on his/her own cluster, a free Parallel SECONDO AMI (Amazon Machine Image) [3] is also published on AWS (Amazon Web Services), with which and the tool *ps-ec2-startInstances* the user can quickly prepare a runnable system on the cluster composed by Amazon EC2 instances.

Due to the length limit of this paper, it is impossible to introduce all auxiliary tools here. However, they are fully explained in plenty of tutorial documents that are published on our website [3] and also some papers that we prepare to publish.

B. Parallel Data Model

At present, queries in Parallel SECONDO are stated in executable language, by which the user can describe the work flow precisely with database objects and operators, in order to achieve the best performance. Correspondingly, the parallel data model is proposed to represent the distributed data and the MapReduce operations in Parallel SECONDO.

Usually, distributed data in Parallel SECONDO are partitioned into a structure named PS-Matrix, in which the content of piece data are stored in sDSs, while the partition schema are kept in the *master database* as so called *flist* objects. For a PS-Matrix, its piece data can be either stored as SECONDO objects in *slave databases*, or exported as disk files in PSFS. The first kind of *flist* is named DLO (Distributed Local Objects) *flist*, and the latter is called DLF (Distributed Local Files) *flist*. The purpose of exporting data into PSFS is to exchange data among DSs directly, because the Mini-SECONDO is able to read a set of PSFS files from the remote DSs, like what Hadoop does in the shuffle stage, and import the files back into the database. Essentially, *flist* is designed as a wrap structure, with which all existing and future SECONDO data types can be distributed and processed in Parallel SECONDO.

Parallel queries are normally formulated with *Hadoop* operators. Each *Hadoop* operator contains a template Hadoop job and an argument function. During the runtime, a Hadoop job is generated based on the template job, and the function query is embedded into its Map or Reduce tasks according to which *Hadoop* operator is used. Within the tasks, the function is processed by Mini-SECONDO in all involved sDSs simultaneously.

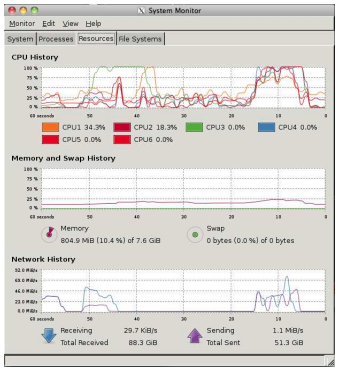

Figure 1: The System Usage of Parallel Processing

For example, in the latter demonstration, the following sequential query makes a hash join between two relations *CityNodesNew* and *CityWays*, based on their *NodeId* and *NodeRef* attributes, respectively.

```
query
  CityNodesNew feed  CityWays feed
  itHashJoin[NodeId, NodeRef]
consume;
```

It can be easily transformed to the parallel statement as:

```
query
  CityNodesNew_NodeId_dlo
  CityWays_NodeRef_dlf
  hadoopReduce2[NodeId, NodeRef, DLF, PS_SCALE
  ; . feed .. itHashJoin[NodeId, NodeRef] ]
collect[] consume;
```

Here both involved relations are first distributed on the cluster as two *flist* objects *CityNodesNew_NodeId_dlo* and *CityWays_NodeRef_dlf*. Normally we recommend the user to name *flist* objects as triples consisting of the original relation name, the partition attribute and the *flist* type. In such a way, it is easier to distinguish the *flist* objects that are created for the same data set, but partitioned in different ways.

Steps	Sequential Processing				Parallel Processing			
	Query Number	Elapsed Time (secs)			Query Number	Elapsed Time (secs)		
		Arnsberg	China	California		Arnsberg	China	California
1	1	241	432	2181	3	150	223	875
2	1	288	583	2750	3	260	260	634
3	4	2192	4077	60170	6	473	444	2280
4	1	422	722	4925	1	381	395	612
5	3	962	2320	9294	5	339	336	845
In total	10	4105	8134	79320	18	1603	1834	5246

Table I: The Comparison of Generating Road Network between SECONDO and Parallel SECONDO

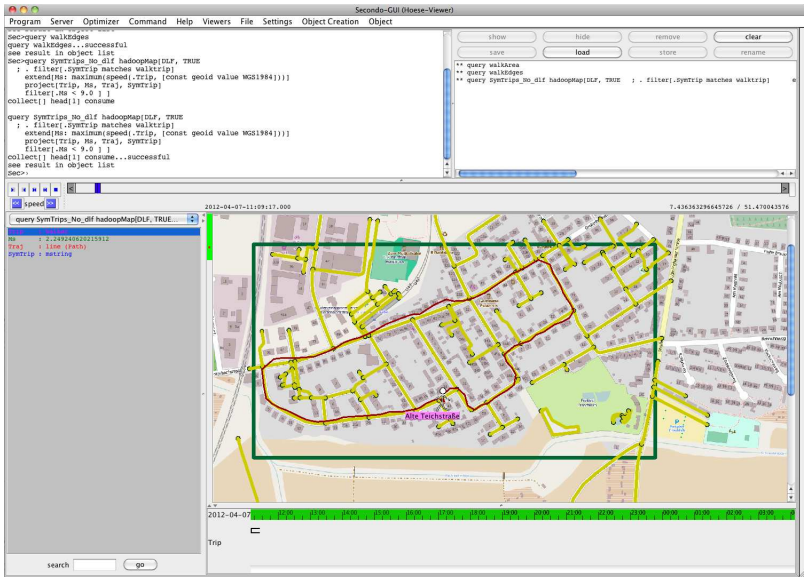

Figure 2: A Screenshot of Symbolic Trajectory Pattern Matching in Parallel SECONDO

The parallel query basically is built up with a *Hadoop* operator **hadoopReduce2**. It takes two input *flist* objects, re-distributes them based on the partition attributes in the Map stage, and then processes the argument function in the Reduce stage. The partition attributes are the first two parameters that the operator accepts, here they are indicated exactly the same as the join attributes since the whole query makes up a reduce-side join operation. Afterwards, *PS_SCALE* reduce tasks are started, processing the argument function on all sDSs at the same time. The *PS_SCALE* is an integer, denoting the size of the reduce tasks, and the argument function is almost the same as the sequential query. At last, this *Hadoop* operator is set to return a DLF *flist*, so that the distributed result can be gathered into the *master database* with the operator **collect**, and then be saved as a normal SECONDO relation by the **consume** operator.

Figure 1 illustrates the system usage during the parallel procedure for this example, by profiling one computer from our testbed with the default system monitor. Obviously the whole procedure is divided into two parts, corresponding to the Map and Reduce stages. All processor cores are fully used during the Reduce stage, and the network is also fully used at the beginning of both stages.

III. DEMONSTRATION

Our demonstration illustrates the practicability of Parallel SECONDO by solving realistic problems. First a road network is generated with the data from OpenStreetMap (OSM) [2] . Next a set of personal geometric trajectories is given, which is observed by GPS devices, and matched to the road network [11]. At last, symbolic trajectories are produced so as to find certain trajectories with semantic patterns [12]. Although all these technologies are far beyond the scope of this paper and studied in single-computer SECONDO, they can still be easily stated and processed in Parallel SECONDO efficiently.

In the process of the demonstration, both the sequential and the parallel solutions are introduced on our cluster. The cluster contains six computers, each has a AMD Phenom(tm) II X6 1055T processor with six cores, 8 GB memory and

two 500 GB hard disks. A Parallel SECONDO is set up on five of them, each computer is set with two DSs. Besides, a single-computer SECONDO is installed on the remaining computer of the cluster. The performances of these two systems are compared with the default system monitor, like the one shown in Figure 1.

The road network generation contains a number of queries that can be roughly divided into the following five steps.

1) Decompose the downloaded OSM data and import them into SECONDO as six relations.
2) Make up a relation named *NewNodes* to get all nodes from an imported relation. Each node is a SECONDO *point* object, containing both its longitude and latitude values. At last, all nodes are sorted according to the Z-order.
3) Save all ways into a relation named *Ways*. Each way indicates a continuous poly-line, containing a set of *line* objects that are assigned with the same *WayId* attribute value. Afterwards, all ways that contain the "highway" tag are extracted to make up a new relation named *Roads*. This tag is defined by OSM, indicating all routes connecting two nodes and being used by motorised vehicles, pedestrians, cyclists, horse riders etc. At last, two R-Tree indices are built upon these two relations.
4) Find all starting, ending and crossing nodes of the Roads, and save them into the relation *Nodes*.
5) Split the roads to edges, while each edge is a piece of road without being cut off by the other edges. In order to indicate one-way roads, each edge is stored as a *sline* object which uses a boolean value to indicate its direction. For a road with two directions, its edges are stored twice with both directions. They are all stored in the relation called *Edges*.

The above queries are time-consuming in single-computer SECONDO since there are a lot of sort and join operations. However, all of them can be converted into corresponding parallel queries and gain considerable speed-up by being processed in Parallel SECONDO. The left part of Table I shows the elapsed time of generating the road network sequentially for three regions: Arnsberg, China and California. The OSM data update all the time, hence till this paper is written, the sizes of the involved data are 1.3GB, 2.3GB and 9.1GB, respectively. The computer comes from either the cluster or one that has a comparable computing power. The same procedure is converted and processed in Parallel SECONDO with all six computers of the cluster, and the performance is shown in the right part of the table. It is clear that Parallel SECONDO usually needs more queries for the same step, but it achieves an impressive speed-up on all steps, especially for large-scale problems.

It is unpractical to perform the complete road network generation during the demonstration period, therefore only several of them are picked out and explained with details, like the one that we introduced in the last section.

In the latter demonstration, we keep using the personal collected trajectories that described in [12], but generating the map-matched and symbolic trajectories all within Parallel SECONDO. The road network is built for the Arnsberg region, and created in advance. The result of one matched trajectory is shown in Figure 2, where the dark green rectangle fences a certain area within the network. Inside that region, roads imported from the OSM data are marked as yellow poly-lines. Apart from that, the red path indicates a trajectory of a person, denoted by the white icon, who took a short walk within twenty minutes inside that road network. The movement of the person can also be animated by the graphical interface. Along with that, the name of the passed street is displayed below the humanoid icon.

IV. ACKNOWLEDGMENT

We are grateful for the research grant provided by AWS in Education, which supports our study in EC2. Besides, the first author is also thankful to the financial support from Chinese Scholarship Council (CSC).

REFERENCES

[1] Hadoop. http://hadoop.apache.org/.

[2] Open Street Map. http://www.openstreetmap.org.

[3] Parallel Secondo. http://dna.fernuni-hagen.de/secondo/ParallelSecondo.

[4] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. *Proc. VLDB Endowment*, 2(1):922–933, 2009.

[5] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Symposium on Operating Systems Design & Implementation - Volume 6*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[6] L. Forlizzi, R.H. Güting, E. Nardelli, and M. Schneider. A Data Model and Data Structures for Moving Objects Databases. *ACM SIGMOD Record*, 29:319–330, 2000.

[7] R.H. Güting, T. Behr, and C. Düntgen. SECONDO: A Platform for Moving Objects Database Research and for Publishing and Integrating Research Implementations. *IEEE Data Eng. Bull.*, 33(2):56–63, 2010.

[8] R.H. Güting, V.T. De Almeida, and Z. Ding. Modeling and Querying Moving Objects in Networks. *The VLDB Journal*, 15(2):165–190, 2006.

[9] C. Lema, J. Antonio, L. Forlizzi, R.H. Güting, E. Nardelli, and M. Schneider. Algorithms for Moving Objects Databases. *The Computer Journal*, 46(6):680, 2003.

[10] Jiamin Lu and Ralf Hartmut Güting. Parallel Secondo: Boosting Database Engines with Hadoop. In *ICPADS*, pages 738–743, 2012.

[11] F Marchal, J Hackney, and Kay W Axhausen. Efficient map matching of large global positioning system data sets: Tests on speed-monitoring experiment in Zürich. *Transportation Research Record: Journal of the Transportation Research Board*, 1935(1):93–100, 2005.

[12] Fabio Valdés, Maria Luisa Damiani, and Ralf Hartmut Güting. Symbolic Trajectories in SECONDO: Pattern Matching and Rewriting. In *DASFAA (2)*, Lecture Notes in Computer Science, pages 450–453. Springer, 2013.