Highly Distributable Associative Memory Based Computational Framework for Parallel Data Processing in Cloud

Amir Hossein Basirat(✉), Asad I. Khan, and Balasubramaniam Srinivasan

Clayton School of IT, Monash University Melbourne, Melbourne, Australia
{Amir.Basirat,Asad.Khan,Bala.Srinivasan}@monash.edu

Abstract. One of the main challenges for large-scale computer clouds dealing with massive real-time data is in coping with the rate at which unprocessed data is being accumulated. In this regard, associative memory concepts open a new pathway for accessing data in a highly distributed environment that will facilitate a parallel-distributed computational model to automatically adapt to the dynamic data environment for optimized performance. With this in mind, this paper targets a new type of data processing approach that will efficiently partition and distribute data for clouds, providing a parallel data access scheme that enables data storage and retrieval by association where data records are treated as patterns; hence, finding overarching relationships among distributed data sets becomes easier for a variety of pattern recognition and data-mining applications. The ability to partition data optimally and automatically will allow elastic scaling of system resources and remove one of the main obstacles in provisioning data centric software-as-a-service in clouds.

Keywords: Associative memory · Neural networks · Big data · MapReduce · Graph Neuron

1 Introduction

While the opportunities for parallelization and distribution of data in clouds have brought some efficiency, existing relational and object-oriented data models in particular, make storage and retrieval processes very complex, especially for massively parallel real-time data. Chaiken et al. [1] observe that the challenge of processing voluminous data sets in a scalable and cost-efficient manner has rendered traditional database solutions prohibitively expensive. At the other end of the spectrum high-performance computing (HPC) has advanced rapidly but dominantly focused on computational complexity and performance improvements. Virtual HPC in the cloud has significant limitations especially when big data is involved. According to Shiers [2], "it is hard to understand how data intensive applications, such as those that exploit today's production grid infrastructures, could achieve adequate performance through the very high-level interfaces that are exposed in clouds". The efficiency of the cloud system in dealing with data intensive applications through parallel processing essentially lies in how data is partitioned and processing is divided among nodes. As a result,

I. Stojmenovic et al. (Eds.): MOBIQUITOUS 2013, LNICST 131, pp. 66–77, 2014.
DOI: 10.1007/978-3-319-11569-6_6

data access schemes are sought to be able to efficiently handle this partitioning automatically and support the collaboration of nodes in a reliable manner. Google's MapReduce [3] and Microsoft Dryad [4] have achieved greater scalability than parallel databases. However this comes at a cost; time-consuming analysis and code customization are required when dealing with complex data inter-dependencies. Moreover, real-time reliability guarantees remain elusive. Conceptually, the approach also suffers from certain key limitations:

In the MapReduce type of query processing, the map tasks are assumed to to be fully independent. Applied to massive relational or object-oriented data, however, large records or objects resulting after aggregation and analytics are themselves often broken into parts and distributed creating dependencies and requiring trade-offs between redundancy (for speed), coherence (for integrity under frequent updates) and compromises to parallel schedulability, as they break assumptions of mutual independence. In practice, MapReduce functions are implemented imperatively and produce numerous intermediary entities - between the map and reduce stages e.g. in the form of intermediate files. In many applications, these files must be sorted and moved around before they are input to the reduce function. This system wide sort and redistribution incurs considerable processing and communication costs and is either fundamentally non-scalable or requires fine-tuned architecture-aware access mechanisms.

While assisting designers and developer s with few predefined architectural patterns [5] for many applications, the MapReduce data flow model is also rigid, limits variation and hence increases the complexities of dealing with errors, fault-tolerance, performance and other end-to-end non-functional issues. Some exploratory research implementations are using key/value pairs with distributed "Spaces" (for instance Java Spaces or other derivatives of Linda tuple spaces [6]) to simplify data sharing and conceptually separate shared data from the computational tasks. However, this simplification comes with significant efficiency loss and exacerbates uncertainty of predicting reliability and real-time behavior.

Hence, MapReduce cannot automatically scale up for many applications and data sets, in practice. Reconciling MapReduce with Associated Memory concepts, in particular for adaptive and fast data access, aggregation and movement will be a key contribution of the proposed technique in this paper. Our proposed scheme preserves the strength of the MapReduce model and eliminates/alleviates most of these constraints in a well-integrated manner where there is no outward change to the way in which MapReduce models are deployed and used. In this context, our proposal will investigate inclusion of an associative approach in the MapReduce model to support application specific pattern recognition and data-mining operation. For efficient analytics, Map functions need to be embedded in streams as it is unrealistic to literally preserve and record all raw data from sensor streams. On the other hand the complexity of some analytics tasks renders them inappropriate for real-time online processing (for example in transport and plant health monitoring). Hence it necessitates a combination of (1) selective stream functions that efficiently query, filter and aggregate information in adaptable ways and, (2) streaming the results into the cloud for later offline processing and analytics.

An associative memory based processing scheme that efficiently performs large-scale data processing will offer a broad spectrum of innovative cloud applications by formatting data universally within the network. It helps alleviate data imbalances by replacing rigid referential data access mechanisms with more distributable associative processing. Hierarchical structures in associative memory models are of interest as these have been shown to improve scalability whilst preserving accuracy in pattern recognition applications [7]. Our proposal is based on a special type of Associative Memory (AM) model, which has been specially designed for distributed processing [8–14] and readily implemented within distributed architectures. Thus our primary aim in this paper is to introduce an access scheme that will enable fast data retrieval across multiple records and data segments associatively, utilizing a parallel approach. Doing so will yield a new form of database-like functionality that can scale up or down over the available infrastructure without interruption or degradation, dynamically and automatically.

2 Graph Neuron for Scalable Recognition

Transforming big data into valuable information requires a fundamental re-think of the way in which future data management models will need to be developed on the Internet. Unlike the existing relational, hierarchical and object-oriented schemes, associative models can analyze data in similar ways to which our brain links information. Such interactions when implemented in voluminous data clouds can assist in searching for overarching relations in complex and highly distributed data sets with speed and accuracy. This proposal improves MapReduce-based cloud applications in a number of different ways by uniformly formatting data in a standard two-dimensional representation. It eliminates data imbalances and completes transition to cloud by replacing referential data access mechanisms with more versatile and distributable associative functions, which allow complex data relations to be easily encoded into the keys as patterns. These patterns can be applied in a variety of applications requiring content recognition e.g. image databases, search within large multimedia files, and data mining. Algorithmic strengths of the MapReduce approach are investigated for the first time in context with the effectiveness of one-shot learning based parallelism provisioned via our distributed pattern recognition approach.

The principle of associative memory based learning will be implemented through the use of hierarchically connected layers, with local feature learning at the lowest layer and upper layers combining features into higher representations. Our approach will entail a two-fold benefit. Applications based on associative computing models will efficiently utilize the underlying hardware that scales up and down the system resources dynamically and automatically, controls data distributions and allocation of the computational resources in the cloud. In order to achieve the aforementioned objectives, an initial step would be to develop a distributed data access scheme that enables record storage and retrieval by association, and thereby circumvents the partitioning issue experienced within referential data access mechanisms. In our model, data records are treated as patterns. As a result, data storage and retrieval can be performed using a distributed pattern recognition approach that is implemented through the integration of loosely-coupled computational networks, followed by a divide-and-distribute approach that facilitates distribution of these networks within the cloud dynamically. Our online-learning associative memory scheme is conceived on the principle that “moving computation is much cheaper than moving data". Hence, it will provide methods for automatic aggregation and partitioning of associated data in the cloud for widely used data sets.

The MapReduce model does not explicitly provide support for processing multiple related heterogeneous datasets. While processing data in relational models is a common requirement, this restriction limits its functionality when dealing with complex and unstructured data such as images. Relational databases use a separate, uniquely-structured table, for each different type of data for specific applications; programmers must know the precise structure of every table and the meaning of every column a priori. To overcome this, we explored possibilities to evolve a novel virtualization scheme that can efficiently partition and distribute data for clouds. For this matter, loosely-coupled associative techniques, not considered so far, can be pivotal to effectively partition and distribute data in the cloud. Our associative model will use a universal structure for all data types. Information about the logical structure of the data – metadata – and the rules that govern it may be stored alongside data. This allows programmers to work at a higher level of abstraction without having to know the structural details of every data item. Hence, our approach to cloud-based data processing is unique. It elevates the MapReduce key-value scheme to a higher level of functionality by replacing the purely quantitative key-value pairs with higher order data structures that will improve parallel processing of data with complex associations (or dependencies). By having an associative key/value framework, we can deal with data in any form and in any representation simply by using a pattern matching model (including fuzziness), which treats data records as patterns and provides a distributed data access scheme that enables balanced data storage and retrieval by association. We believe that the performance of MapReduce parallelism as a scalable scheme for data processing in clouds may be significantly improved by transforming the data processing operations into one-shot distributed pattern matching sub-tasks, which in distributed computations are performed in-network, enabling data storage and retrieval by association (instead of pre-set referential data access mechanisms).

2.1 Graph Neuron (GN)

Graph Neuron (GN) [12] is an associative memory algorithm, which implements a scalable AM device through its parallel in-network processing framework. Associative memory architecture differs from conventional memory architecture in the sense that the store and recall operations on memory contents are based on the association with input value rather than based on the address of the memory content. Hence, associative memory-based pattern recognition algorithms are able to offer high recognition accuracy as compared to other algorithms which implement recognition using conventional memory architecture. In addition to its associative memory architecture, GN also follows some characteristics of graph-based pattern recognition algorithms [8]. However, GN implements in-network processing that solves the scalability issue (computationally prohibitive against an increase in the size and database of patterns) in other graph-based pattern recognition algorithms [9].

GN recognition process involves the memorization of adjacency information obtained from the edges of the graph. Adjacency information for each GN is represented using the (left, right) formation. Each activated GN therefore records the information retrieved from its adjacent left or right nodes as illustrated in Fig. 1. In the GN terminology, this adjacency information is known as bias entry where each GN maintains an array of such entries. The entries for the entire stored pattern are collectively stored in the bias arrays. Each GN would hold a single bias array containing all the bias entries obtained in the recognition processes. In this context, GN offers low storage complexity in recognition process since each GN is only required to store a single array. Furthermore, each GN's bias array only stores the unique adjacency information derived from the input patterns.

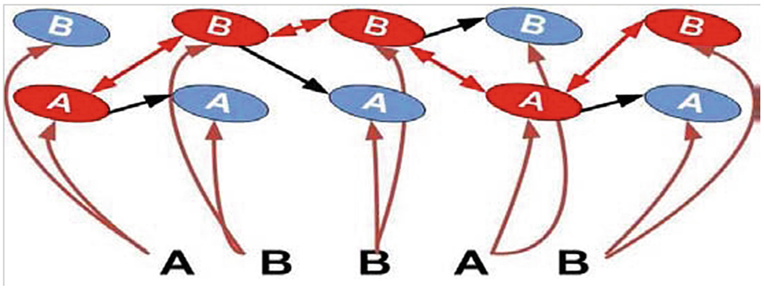

Fig. 1. GN activation from input pattern "ABBAB"

GN's limited perspective on overall pattern information would affect a significant inaccuracy in its recognition scheme. As the size of the pattern increases, it is more difficult for a GN network to obtain an overview of the pattern's composition. This produces incomplete results, where different patterns having similar sub-pattern structure leads to false recall. The limited perspective of GNs, owing to purely adjacency based computations, was widened through the Hierarchical Graph Neuron (HGN) approach [11].

2.2 Hierarchical Graph Neuron (HGN)

In order to solve the issue of the crosstalk due to the limited perspective of GNs, the capabilities of perceiving GN neighbors in each GN is expanded in Hierarchical Graph Neuron (HGN) to prevent pattern interference. The underlying principle of HGN implementation is such that the capability of "perceiving neighbors" in each GN within the network must be expanded. This is achieved by having higher layers of GN neurons that oversee the entire pattern information. Hence, it will provide a bird's eye view of the overall pattern. HGN extends the functionalities of GN algorithm for pattern recognition by providing a bird's eye view of the overall pattern structure. It thus, eliminates the possibility of false recalls in the recognition process.

3 Edge-Detecting Hierarchical Graph Neuron (EdgeHGN)

An important aspect in the development of pattern recognition scheme is its algorithmic design. A proper design will lead to high efficiency and has the ability to generate a more accurate classification strategy. Graph Neuron based algorithms have been developed based upon two different concepts known as graph-matching and associative memory. These two concepts have given an added advantage in terms of scalability for GN-based algorithm implementations. GN has the ability to perform pattern recognition processes on distributed systems due to its simple recognition procedure and lightweight algorithm. Furthermore, GN incurs low computational and communication costs when deployed in a distributed system. Previous parts of this paper have analyzed GN and HGN. In this section, the algorithmic design of a newly proposed Edge Detecting Hierarchical Graph Neuron (EdgeHGN) algorithm for distributed pattern recognition scheme for large-scale data sets is presented. The proposed approach extends the scalability of the existing Hierarchical Graph Neuron (HGN) implementation by reducing its computational requirements in terms of the number of neurons for recognition processes while providing comparable recognition accuracy as HGN implementation. EdgeHGN provides a capability for recognition process to be deployed as a composition of sub-processes that are being executed in parallel across a distributed network. Each sub-process is conducted independently from each other, making it less cohesive as compared to other pattern recognition approaches.

3.1 EdgeHGN Architecture

In our proposed novel EdgeHGN model, we reduce redundant data content for recognition by applying a Drop-Fall algorithm on the input pattern. This results in lesser number of processing neurons which in turn results in lower communication overhead within the scheme. The dividing path produced by Drop-fall algorithm depends on three aspects: a start point, movement rules, and direction. In our approach, a drop-fall scheme will be applied to the pattern which ensures producing the least number of neurons. By applying a simple drop-fall algorithm, we can reduce number of redundant processing neurons in the binary character image while maintaining all character data bits. This approach is shown in Fig. 2 where a Descending-left drop-fall algorithm is applied on the input pattern reducing number of processing nodes for each EdgeHGN subnet significantly (total number of GN nodes are decreased from 49 to 39 in this example). This reduction will not only minimize communication costs but also having an edge detection feature within the scheme can improve recognition accuracy to a high degree. Furthermore, lesser number of neurons results in lower response time which is of high interest for real-time pattern matching problems. EdgeHGN adds a clustering mechanism in pattern recognition by dividing and distributing patterns into sub-patterns. Each of the sub-patterns undergoes a one-shot recognition procedure. The results of sub-recognition will cumulatively add up to obtain the actual recognition result. Each processing node in clustered EdgeHGN configuration may perform recognition on each sub-pattern independently from other processing nodes.

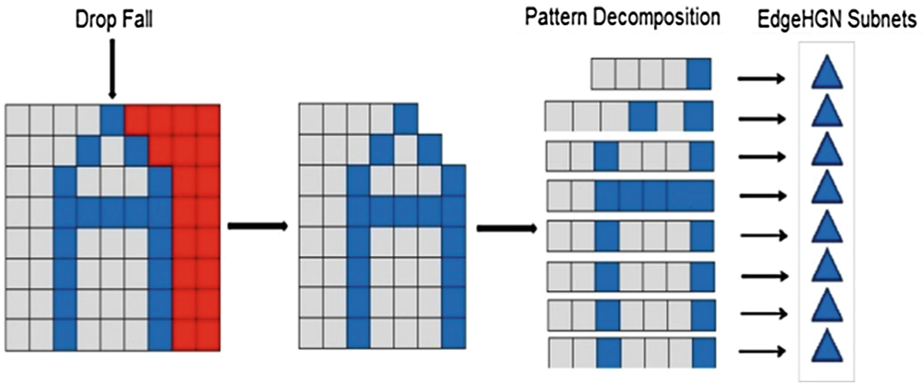

Fig. 2. EdgeHGN progressively removes unnecessary nodes from the two dimensional data representation.

This configuration is intended to be used on coarse-grained networks such as grid and cloud computing, in which additional processing and storage capacity made available to be used. An important benefit of having this EdgeHGN cluster performed on a single processing node is such that it eliminates all the communication actions involved in EdgeHGN message passing model for distributed systems. For each sub-pattern recognition process, each node only communicates back the corresponding index generated, therefore reducing the chances of recognition failures due to transmission or communication errors.

3.2 EdgeHGN Subnet Communication Scheme

In EdgeHGN implementation, after applying drop-fall scheme on the input pattern and removing redundant processing neurons, we will form EdgeHGN subnets. In EdgeHGN implementation, the core recognition process is conducted at the sub-pattern level. There are four stages involved in this process for each EdgeHGN subnet:

Stage 1. After receiving an input, each activated GN at the base layer will send a signal message to other nodes in the adjacent columns containing the row number/address of the activated node. Those activated nodes that are at an edge of the layer will only send the activation signal messages to the GNs in the penultimate columns. The activated GNs that receive the signal messages from their adjacent neighbors will respond by updating their bias array noting the activation signals. All other GNs will remain inactive.

Stage 2. All active GNs at the base layer will then update their bias arrays. If the bias entry value, received from both the activated nodes in proceeding and succeeding columns have been recorded, the index of the entry will be sent to the respective GN in the same position at the higher layer. If the value is not found within the bias array, then a new index will be created and sent to the GN node in the higher layer. Note that active nodes at the edges of the base layer will not be communicating with higher layer nodes since there is no node present at the edges of the higher layer owing to the pyramid-like structure of the EdgeHGN subnets.

Stage 3. GN nodes at a layer above the base that receive a signal message, containing the index of the bias entry that has been created or recalled from stage 2, will be activated. Similar process as in stages 1 and 2 will occur. However, the contents of the signal messages from preceding and succeeding columns would be in the form of (*left, middle, right*) for non-edge nodes and either (*left, middle*) or (*middle, right*) for the edge nodes. The values for left, middle, and right are derived from the indices retrieved from the lower layer nodes. After the message communication between adjacent nodes has completed, the active GNs will update their bias arrays and send the stored/recalled index/indices to the node at the same position in the higher layer (except for the GNs at the edges). This stage will be repeated for each layer above the base layer, until it reaches the top layer GN nodes.

Stage 4. One of the top layer GNs will receive a bias index from a GN in the layer underneath it. This top layer activated node will search its bias array for this index. If the index is found, then this node will trigger a recall flag with the recalled index. Otherwise, it will trigger a store flag and store the new index in its bias array. The signal message sent by the top layer active GN marks the completion of the recognition at sub-pattern level.

3.3 EdgeHGN Communication Complexities

Communications in the EdgeHGN recognition scheme involve a message-passing mechanism, in which a single processing node communicates with other nodes in the network for exchanging messages. It is composed of two different types, namely macro- and micro-communication. In *macro-communication*, communication costs at system level are taken into account, i.e. communications incurred between SI Module and EdgeHGN subnets. On the other hand, *micro-communication* deals with GN communications within a particular subnet for each pattern introduced into the system.

EdgeHGN Macro-Communications. Macro-communication in EdgeHGN implementations happens between SI Module node and either base layer GNs or top GNs in each subnet. It occurs at three different phases:

Network generation phase: SI module is responsible for communicating possible input values of the patterns, which will be used in the recognition process to all base layer GNs within EdgeHGN subnets.

$$n^{msg}_{SI \rightarrow sub} = n_{sub} \times S_{sub} \times v$$

Pattern input phase: SI module decomposes pattern into a number of sub-patterns according to the number of subnets available. Consequently, these sub-patterns will be sent to each subnet within the network. However, in the actual format, SI module will communicate directly with each GN at the base layer of each EdgeHGN subnet. Hence, the number of messages communicated is similar to the number of messages in network generation phase

$$n^{msg}_{SI\to sub} = n_{sub} \times S_{sub} \times v$$

Result communication phase: After recognition process in each EdgeHGN subnet is completed, the results (in terms of recall or store) will be communicated back to SI module for further analysis. In regards to the communication cost, the total number of messages communicated from subnets to SI module is equivalent to the number of subnets available:

$$n^{msg}_{sub\to SI} = n_{sub}$$

EdgeHGN Micro-Communications. In terms of micro-communications, we have communications among GNs within the base layer. For each GN in the base layer, the amount of message communications incurred could be derived from the number of messages communicated between adjacent neurons for each input sub-pattern.

Base Layer: For GNs at the edge of base layer, the number of communication exchange is equivalent to the number of different elements within the sub-pattern. For non-edge GNs, the communication is required between adjacent neurons in both the preceding and the succeeding columns as well as the communication of bias indices to the GNs at the next higher layer. In this context, the amount of message exchange is $v^2 + 1$.

$$n^{msg}_{l_{base}} = \left((v^2+1)(S_{sub} - 2) + 2v\right)$$

Middle layers: The communication costs for GNs in the middle layers are similar to that at the base layer. However, the difference would be in the number of nodes available within each layer. For each middle layer i, where $1 \le i \le top-1$, the number of message exchanges occurred for single input sub-pattern recognition could be derived as the following:

$$n^{msg}_{l_i} = \left((v^2+1)(S_{sub} - (2i+2)) + 2v\right)$$

$$n^{msg}_{l_{total\,i}} = \sum_{i=1}^{top-1} \left((v^2+1)(S_{sub} - (2i+2)) + 2v\right)$$

Top layer: These GN nodes are only responsible for communicating the final index for each sub-pattern stored/recalled to the SI module. The costs for communicating these indices have been included in the macro-communication evaluation.

4 Simulation and Results

Hadoop can be set-up and configured in 3 different modes. Standalone or local mode is where no Hadoop daemons running and everything runs in a single JVM. In Pseudo-distributed mode, Hadoop daemons run on the local machine, thus simulating a cluster on a small scale. And in a fully distributed mode the Hadoop daemons run on a cluster of machines. For our performance benchmarks, a Pseudo-distributed mode Hadoop environment is set-up with default configuration settings but some changes are made to the settings to gain better performance, e.g. the max data chunk size is set to 256 MB instead of 64 and heap size for task executer JVM is increased to 512 MB for better memory allocation and garbage collection. In addition, the performance of Hadoop MR and newly proposed EdgeHGN based MR against is compared against one of the commonly used parallel database management systems called Vertica. The Vertica database is a parallel DBMS designed for large data warehouses. The main distinction of Vertica from other DBMSs is that all data is stored as columns, rather than rows. In Fig. 3, we can see performance of all three schemes while performing a simple task of a pattern search. In Vertica, a pattern search for a particular field is simply running a query in SQL which requires a full table scan:

$$SELECT * FROM\ Data\ WHERE\ field\ LIKE\ \text{‘\%XYZ\%’};$$

On the other hand, the MR program consists of just a Map function that is given a single record already split into the appropriate *key/value* pair and then performs a sub-string match on the value. If the search pattern is found, the Map function simply outputs the input key/value pair to HDFS. Because no Reduce function is defined, the output generated by each Map instance is the final output of the program. As clearly shown here, distributed MapReduce and EdgeHGN based MapReduce perform equally well. For some data input splits EdgeHGN even responds sooner in time and average response time looks better. Vertica performs the best here as we simply run a very single query against the database.

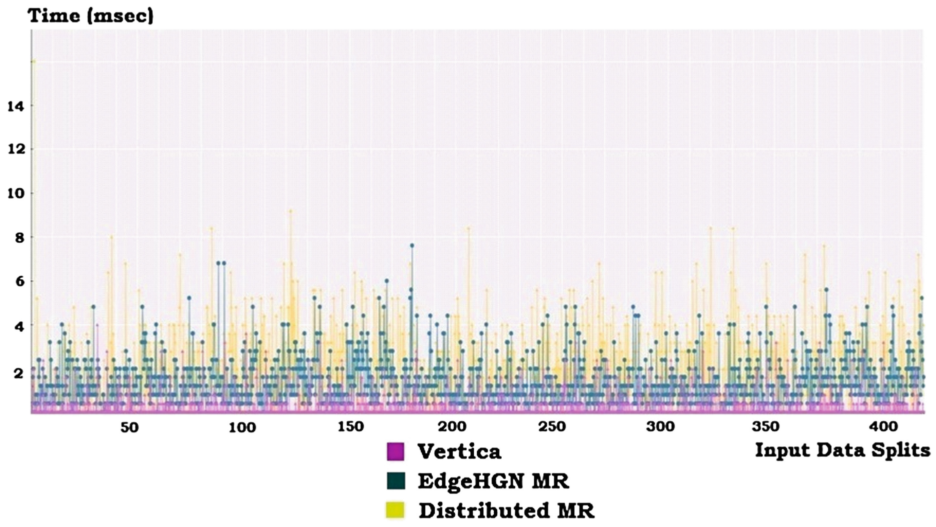

Fig. 3. Comparing Distributed MapReduce, EdgeHGN based MapReduce and Vertica, performing alphanumeric pattern search on input data splits of 256 MB in size.

One of the reasons that Hadoop performance and EdgeHGN performance are lower compared with Vertica is the fact that we are running both in a Pseudo-distributed mode and not in a fully distributed mode in a cluster where memory is allocated to the process independently. The other reason is that considering the limited number of data chunks that we process in both Pseudo distributed Hadoop and EdgeHGN based MR, Hadoop's start-up costs can become the limiting factor in its performance. In fact for small queries, Hadoop startup costs can dominate the execution time. In our observations, we found that it can take 10–20 sec before all Map tasks have been started and are running at full speed.

5 Conclusion and Remarks

Existing cloud frameworks such as Hadoop MapReduce involve isolating low-level operations within an application for data distribution and partitioning. This limits their applicability to many applications with complex data dependency considerations. This paper explored new methods of partitioning and distributing data in the cloud by fundamentally re-thinking the way in which future data management models will need to be developed on the Internet. Loosely-coupled associative computing techniques, which have so far not been considered, can provide the break through needed for a distributed data management scheme. Using a novel lightweight associative memory algorithm known as Edge Detecting Hierarchical Graph Neuron (EdgeHGN), data retrieval/processing can be modeled as a pattern recognition problem, conducted across multiple records within a single-cycle, utilizing a parallel approach. The proposed model envisions a distributed data management scheme for large-scale data processing and database updating that is capable of providing scalable real-time recognition and processing with high accuracy while being able to maintain low computational cost in its function.

References

1. Chaiken, R., Jenkins, B., Larson, P.A., Ramsey, B., Shakib, D., Weaver, S., Zhou, J.: SCOPE: easy and efficient parallel processing of massive data sets. In: Proceedings of Very Large Database Systems (VLDB), vol. 1(2), pp. 1265−1276 (2008)
2. Shiers, J.: Grid today, clouds on the horizon. Comput. Phys. Commun. **180**, 559–563 (2009)
3. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: OSDI'04: Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation, Berkeley, CA, USA (2004)
4. Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D.: Dryad: distributed data-parallel programs from sequential building blocks. In: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, pp. 59−72, New York, USA (2007)
5. Gamma, E., Helm, E., Johnson, R., et al.: Design Patterns: Elements of Reusable Object-oriented Software. Addison Wesley, Reading (1995)
6. Gelernter, D., Carriero, N.: Generative communication in Iinda. ACM Trans. Program. Lang. Syst. **7**(1), 80–112 (1985)

7. Ohkuma, K.: A hierarchical associative memory consisting of multi-layer associative modules. In: Proceedings of 1993 International Joint Conference on Neural Networks (IJCNN'93), Nagoya, Japan (1993)
8. Muhamad Amin, A.H., Khan, A.I.: Commodity-grid based distributed pattern recognition framework. In: 6th Australasian Symposium on Grid Computing and e-Research (AUSGRID 2008), Wollongong, NSW, Australia (2008)
9. Khan, A.I., Amin, A.H.M.: One shot associative memory method for distorted pattern recognition. In: Orgun, M.A., Thornton, J. (eds.) AI 2007. LNCS (LNAI), vol. 4830, pp. 705–709. Springer, Heidelberg (2007)
10. Baig, Z.A., Baqer, M., Khan, A.I.: A pattern recognition scheme for distributed denial of service (DDOS) attacks in wireless sensor networks. In: Proceedings of the 18th International Conference on Pattern Recognition (2006)
11. Nasution, B.B., Khan, A.I.: A hierarchical graph neuron scheme for real-time pattern recognition. IEEE Trans. Neural Netw. **19**, 212–229 (2008)
12. Khan, A.I.: A peer-to-peer associative memory network for intelligent information systems. In: Proceedings of the 13th Australasian Conference on Information Systems, vol. 1 (2002)
13. Baqer, M., Khan, A.I.: Energy-efficient pattern recognition for wireless sensor networks. In: Mobile Intelligence, pp. 627−659. John Wiley and Sons Inc., Hoboken (2010)
14. Khan, A.I., Muhamad Amin, A.H.: Integrating sensory data within a structural analysis grid. In: Topping, B.H.V., Iványi, P. (eds.) Parallel, Distributed and Grid Computing for Engineering. Saxe-Coburg Publications, Kippen (2009)
15. Catterall, E., Van Laerhoven, K., Strohbach, M.: Self-organization in ad hoc sensor networks: an empirical study. In: ICAL 2003: Proceedings of the Eighth International Conference on Artificial life, pp. 260–263. MIT Press, Cambridge