Agent-Based Parallelization of a Multi-Dimensional Semantic Database Model

1st Alex Li
Division of Computing and Software Systems
University of Washington Bothell
Bothell, WA 98011
0000-0002-4440-2082

2nd Munehiro Fukuda
Division of Computing and Software Systems
University of Washington Bothell
Bothell, WA 98011
0000-0001-7285-2569

Abstract—**Responses to database queries that may be even identical should vary if they are given under a different user context. For instance, queries for wild animals in the context of the ocean versus mountains should be different. Announced in 1993 [1], Mathematical Model of Meaning (MMM) provides users with capabilities to extract data items tightly coupled under different semantic spaces. Such a space is created dynamically with user-defined impression words to compute semantic equivalence and similarity between data items. MMM computes semantic correlations between the key and other data items to achieve dynamic data querying. However, a semantic space creation and a data correlative calculation are computationally demanding. We consider MMM as a practical database application of multi-agent technologies, construct a space over a cluster system, and have multi-agents explore for a given target and its surrounding data items. We use the Multi-Agent Spatial Simulation (MASS) library to implement an agent-based semantic database system and to measure its parallel execution. Compared to a sequential MMM implementation, MASS-based parallelization yielded a 22-time speedup when creating a space, mainly achieved with matrix multiplication. MASS also reduced the time required for distance sorting of multidimensional vectors by 23.7%.**

Index Terms—**agent-based modeling, mathematical model of meaning, semantic database.**

I. Introduction

If big-data computing needs to handle a structured dataset such as a graph or a multi-dimensional space, it makes sense to keep its structure in memory and dispatch agents into it rather than decompose the structure and stream its data to conventional analytic tools such as Spark[1] and Flink[2]. As we previously proposed this agent-based analysis of structured datasets [2], this paper intends to demonstrate agents' applicability to a practical information-retrieval system that deals with high-dimensional semantic spaces.

We consider Mathematical Model of Meaning (MMM) [3] that dynamically constructs a multi-dimensional semantic space from user-given keywords and that responds to database queries in the context of the semantic space. To facilitate the space creation, MMM associates each data item with features; defines interconnections among multiple items in a correlation matrix; and extracts a user-defined semantic space by decomposing the matrix into an eigenvector and performing Fourier expansion between user-provided impression words and the eigenvectors. Given such a semantic space, MMM calculates the Euclidean distance between a query data and all the other items, which sorts out the closet data as the response.

MMM has two computational aspects: (1) multi-dimensional space construction and (2) computational demands in matrix operations and distance calculations. We believe that agent-based data analysis is potential to support these two computations. For a verification, we use the MASS (Multi-Agent Spatial Simulation) library[3] to map semantic spaces over a cluster system with *Places* objects and to move *Agents* objects over the spaces for parallel computing. In particular, we walk agents over a matrix, based on Cannon's algorithm [4] and propagate agents over a semantic space from a test data in search for its closet data items.

This paper's contributions are two-fold: (1) verifying a practical use of agent behaviors (such as agent march and propagation) as data-retrieving techniques applicable to a semantic database and (2) demonstrating performance gains through parallel computation among multi-agents, (i.e., a 22-time speed-up in a space creation and a 23.7% reduction of time required for database queries, both compared to the corresponding sequential execution).

The rest of this paper is organized as follows: section 2 overviews MMM as an application we focus on; section 3 explains our agent-based parallelization techniques for semantic space computation and data retrieval; section 4 demonstrates the performance of agent-based parallelization; section 5 differentiates our MASS-parallelized semantic database from other related systems; and section 6 concludes our discussions.

II. Computational Basis

This section intends to summarize MMM [3]'s computational basis that is needed for its parallelization with multi-agents.

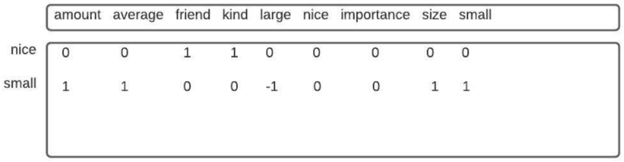
Traditional SQL-like database systems identify queried data through pattern matching. SELECT and WHERE clauses are used to filter data items through user-provided conditions.

This research was partially funded by University of Washington Bothell, Computing and Software Systems Division's RA-ship support.

[1] https://spark.apache.org/

[2] https://flink.apache.org/

[3] https://depts.washington.edu/dslab/MASS



	amount	average	friend	kind	large	nice	importance	size	small
nice	0	0	1	1	0	0	0	0	0
small	1	1	0	0	-1	0	0	1	1

Fig. 1. A matrix to associate each data item with different features

They generally ignore the semantic meaning of data items. Comparing with them, MMM allows users to query data items through semantic associative searches where they are projected onto semantic spaces. For this purpose, MMM takes the following two steps: *step 1* constructs a semantic space and *step 2* computes the distance between data items in a given space.

A. Semantic Space Construction

A space construction gets started with formalizing an input dataset into matrix $A[m][n]$ where all m data items are listed in the row and their n features are distinguished in the column. For instance, entries in a given English dictionary and their explanatory words are considered as data items and their features [5]. A more specific example is Random House Roget's Thesaurus that can be defined as $A[11,000][2,000,000]$ with 11K entries and 2M synonyms/antonyms [6]. As shown in Figure 1, each data item indicates the degree of its association with a different feature, (e.g., 1, 0, and -1 as a positive, a neutral, and a negative association).

From matrix $A[m][n]$, MMM computes the Frobenius norm for each row, (i.e., data) with $\parallel F \parallel = \sqrt{\sum_{i,j} abs(A[i][j])^2}$; normalizes $A[m][n]$ by the vector of $\parallel F \parallel$; and multiplies it with its transpose matrix A^T, which obtains the final correlation matrix A^TA [3]. Its computational complexity corresponds to $O(m^3)$.

MMM then decomposes the correlation matrix A^TA into eigenvectors v that satisfies $A^TA\mathbf{v} = \lambda\mathbf{v}$. A user defines a semantic context as a set of impression words $(d_0, d_1, ..., d_{l-1})$ where $l < m$, which is thus a subset of data items in the original matrix A. For each impression word d_i where $0 \leq i < l$, MMM computes the Fourier expansion between d_i's features $A[d_i][0] \sim A[d_i][n-1]$ and eigenvector $\mathbf{v}$, which results in $\mathbf{d_i}$. This is the impression word d_i's coordinates in the semantic space [3].

B. Data Retrieval

When retrieving user-desirable data items from a given semantic space, MMM needs to calculate all the Euclidean distances between any pair of data items x and y with $\sqrt{\sum_{i=0}^{n} C_i(x[i]-y[i])^2}$ where $\mathbf{x}$ and $\mathbf{y}$ are coordinates in a semantic space. C_i is the weight i of MMM's so-called semantic center. It is computed as an average of all impression words' coordinates in the space [3]. This distance calculation is bound to $O(m^2)$.

MMM considers two data retrieval scenarios [5]. The first is to identify the nearest neighbors from a given keyword P in the semantic space created with impression words d_i. This corresponds to finding the data items closet to P in terms of meaning. The second is to find top N data items from the user-defined semantic space, where users provide no keywords to query against. MMM sorts out top N items closer to the collection of impression words d_i that constitutes a semantic space.

In summary, given m data items, MMM computational complexity is bound to $O(m^3)$ in a semantic space creation, (mostly in matrix multiplication) and $O(m^2)$ in each database query, (i.e., data-to-data distance calculation). Its computational space can be framed in multi-dimensional arrays.

III. Agent-based Parallelization

MMM's input matrix A and correlation matrix A^TA are proportional to the number of data items and their features. Furthermore, any additions or deletions of data items need a $O(m^3)$ re-computation of the correlation matrix. This is our motivation to maintain these matrices in distributed cache. On the other hand, MMM's semantic space is much lighter in its creation as the number of user-provided impression words is limited and the correlation matrix's eigenvectors are ready to use. However, the problem is repetitive queries given to a semantic space, which currently needs $O(m^2)$ Euclidean distance computation. Instead, our strategy injects an agent into a semantic space and propagates its clones until encountering closer data items.

We use the MASS library that instantiates a multi-dimensional distributed array with *Places* and populates agents over the array with *Agents*. *Places* can represent a multi-dimensional semantic database and facilitate parallel function calls of and data exchanges among array elements, (i.e., data items), respectively through *callAll()* and *exchangeAll()*. On the other hand, *Agents* can ramble over the database as queries. They schedule their next search behaviors with *callAll()* and to commit the actions with *manageAll()*. All MASS built-in functions are executed in parallel over cluster computing nodes, each with multi-cores and thus multithreaded. The following two sub-sections give the details of MASS-parallelized MMM.

A. Semantic Space Creation

As described in Section II-A, MMM mainly takes two steps to be ready for answering any semantic database queries: *(step 1)* a semantic space creation - further divided into sub-steps *(a)* matrix multiplication and *(b)* eigenvalue decomposition; and *(step 2)* Euclidean distance computations and data sorting. Based on MMM's computational examples in [1], we first implemented MMM as a sequential Java program from scratch. Figure 2 shows its execution time required for answering the very first query on various datasets, each generated with random double values.

Overall, *step 1a* costs the most computation time in MMM. The matrix normalization and multiplication take over 90% of the entire execution. The slowest is when an input matrix A

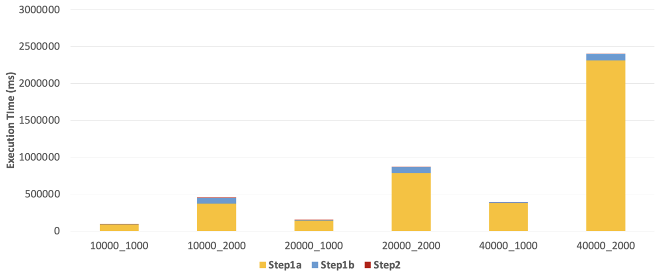


Fig. 2. Sequential MMM execution

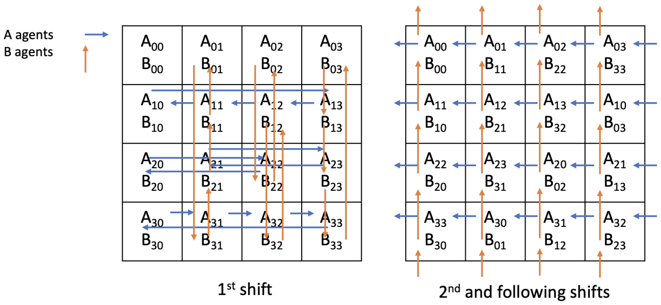


Fig. 3. Cannon's algorithm [4]

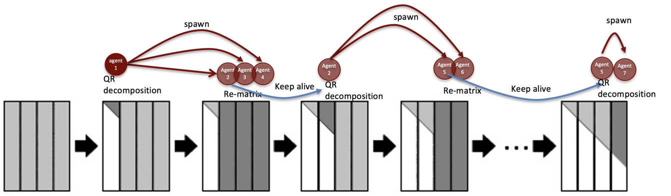


Fig. 4. Block householder transformation for parallel QR factorization [7]

consists of 40,000 data items × 2,000 features, which takes over 40 minutes to complete the calculation. *Step 1b* occupies 7% of the total time on average. The largest overheads were observed when an input matrix A consists of 10,000 data items × 2,000 features, which is 17%. *Step 2* incurs negligible overheads as compared to *steps 1a* and *1b* while its overheads are repetitive as numerous users issue database queries, which we will address in Section III-B.

Given the above measurements, we consider the following agent-based parallelization of semantic space creation.

1) **Matrix multiplication:** uses agents to implement Cannon's parallelization [4]. As shown in Figure 3, this algorithm takes shift 1 and repetitive shift 2. Shift 1 aligns matrices A and B so that processor $P_{i,j}$ gets A_{ix} and B_{xj}. Listing 1 describes agent migration in shift 1, where agents with an even ID (line 4) will pick up matrix A from the current place (lines 6-8), computes the left place index (line 9), and migrate there (line 17), while agents with an odd ID will take care of matrix B (lines 10-16). Thereafter, shift 2 moves each sub-matrix A one step left and each sub-matrix B one step up. Upon each shift, processor $P_{i,j}$ performs block multiplication. We populate A and B agents, each repetitively performing a pair of block shift and multiplication.
2) **Parallel eigen-decomposition:** might consider the block householder transformation for parallel QR factorization [7] where agents perform column-by-column pipelined computation of QR decomposition on column i and matrix re-transformation on columns $i+1 \sim n-1$, as shown in Figure 4. However, the factorization sequentially sweeps a given matrix columns from left to right. Therefore, this parallelization strategy couldn't substantially reduce 17% overheads of the entire computation.
3) **Parallel initialization of input matrix:** uses the MASS library's parallel file I/O that partitions a big file into smaller chunks, each loaded from a different computing node into the respective stripe of *Places* objects. Despite that MMM's input matrix A is described in a CSV format and thus each line length of the file is variable, MASS equally partitions a given file by the number of computing nodes. Therefore, the MASS library may split a data item in its middle. To solve this problem, we add paddings on each line to make sure that all lines of a CSV file have the same length.

Listing 1. Agent migration over matrices

```
1  int row = place.getIndex( )[0]; // retrieve the current place's
2  int col = place.getIndex( )[1]; // indexes: (row, col)
3  int size = place.getSize( )[0];
4  if ( agentAgentId() % 2 == 0 ) {
5    // I'm an A agent to carry matrix A along the row
6    System.arraycopy( this.matrixToCarry, 0,
7                      place.matrixA, 0, place.matrixA.length );
8    place.matrixA = null; // I'll take matrix A away
9    col = ( row != 0 ) ? ( size + col − row ) % size : col;
10 } else {
11   // I'm a B agent to carry matrix B along the column
12   System.arraycopy( this.matrixToCarry, 0,
13                     place.matrixB, 0, place.matrixB.length );
14   place.matrixB = null; // I'll take matrix B away
15   row = ( row != 0 ) ? ( size + row − col ) % size : row;
16 }
17 migrate( col, row ); // ready to migrate
```

B. Data Retrieval

Our measurement found that the slowest response time in one data-retrieval operation (including Euclidean distance calculation and data sorting) was only 1,607 milliseconds. However, as the number of data items gets increased from 10,000 to 20,000, this operation slows down by 1.6 times. An increase of 1,000 to 2,000 features even shows a five-time performance deterioration. Since data retrieval is repetitive and main operations in databases, we parallelize data retrieval using agent propagation, as shown in Figure 5.

Contrary to MMM's exhaustive point-to-point Euclidean distance calculation (Figure 5-(a)), we start propagating agents from the location of a test data item over a given semantic space. The more data items in the space, the quicker agents collide with any other data items (Figure 5-(b)). Of importance

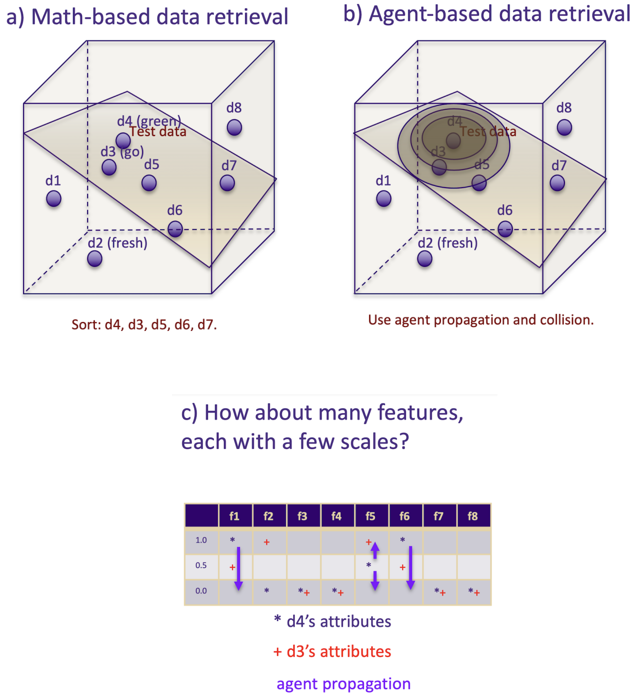

a) Math-based data retrieval
b) Agent-based data retrieval
d1
d2 (fresh)
d3 (go)
d4 (green)
Test data
d5
d6
d7
d8
d1
d2 (fresh)
d3
d4
Test data
d5
d6
d7
d8
Sort: d4, d3, d5, d6, d7.
Use agent propagation and collision.
c) How about many features, each with a few scales?
f1
f2
f3
f4
f5
f6
f7
f8
1.0
0.5
0.0
* d4's attributes
+ d3's attributes
agent propagation

Fig. 5. Agent propagation over a semantic space

is how to implement such agent propagation in a higher-dimensional space. Figure 5-(c) describes how to address the problem. The agent propagation takes the following four steps:

1) Create $places[s][f]$ where $f = \#features$ and $s = \#scales$ per feature. Figure 5-(c) shows a creation of $places[3][8]$, as data items are associated with eight features, each distinguishing its intensity with scales 0.0, 0.5, and 1.0, thus with three scales.
2) Populate agents, each starting from $places[i][j]$ where a test data has feature j with its scale value i. Note that an agent does not have to be spawned at every single feature j but can cover a group of features k through to l where $0 \leq k < l < f$ for the purpose of saving $\#agents$ to spawn. In Listing 2, $feature\_min$ and $feature\_max$ corresponds to k and l, respectively.
3) Move agents vertically from one scale to another on s. Listing 2 illustrates their migration algorithm. Upon encountering a new data item (lines 3-5), an agent calculates its distance from the test data (line 6). However, we should let the agent react to only data items within the range from $m/\#agents \times agentID$ to $m/\#agents \times (agentID + 1)$, where $m = \#data\ items$. This migration and computation scheme prevents multiple agents from discovering the same data item. Thereafter, the agent migrates to a next scale (lines 10-12).
4) Stop agent propagation when agents complete visiting all $places[s][f]$ or find a given number of top data items.

Listing 2. Distance calculation and agent migration

```
 1 // this = this agent
 2 for( int i = this.feature_min; i < this.feature_max; i++ ) {
 3   if( valueMap.containsKey( i ) ) { // encounter a new data item
 4     double[] value =
 5       place.getFromRandomAccessFile(datafile, i, this.dataSize);
 6     double distance = calculateDistance( place.keyData, value );
 7     this.results.put( i, distance );
 8   }
 9 }
10 int newY = currY;
11 int newX = (currX + 1) % this.levels;
12 migrate( newY, newX ); // move horizontally along feature Y
```

IV. Parallel Performance

We conducted experiments on top of a Linux cluster at the University of Washington Bothell. The system has eight computing nodes, each with 4-core Intel(R) Xeon(R) Gold 6130 CPU and 20GB memory space. The Java version of MASS library was configured with 4GB initial heap size and 12GB maximum heap space. The testing dataset was randomly generated in double precision.

A. Semantic Space Creation

Figure 6 visualizes the execution time of matrix multiplication. The matrix size is $2,048 \times 2,048$. We include time elapsed for MASS initialization, *Places* initialization, file read operations, and actual calculation. MASS sets up a cluster system with MASS.init(). During this initialization stage, MASS establishes an SSH connection between the master node and each worker node, and exchanges messages between the nodes to verify their connections. The *Places* initialization time includes their range partition over computing nodes and their instantiation on different computing nodes.

As compared to the sequential execution that needs 115,694 milliseconds, MASS achieved the entire execution within 20,371 and 20,588 milliseconds, using or without using parallel I/O respectively. The computation time itself takes only 5,223 milliseconds, using four computing nodes, which reduces 95% of the sequential execution (or runs 22 times faster). This is because MASS not only facilitates distributed memory space but also has a multi-core capability, thus fully using 4×4 CPU cores, which achieves super-linear improvements in its execution performance.

The main problem in MASS is its SSH and *Places* initialization. The overheads do not drastically change regardless of the number of computing nodes while we can still expect further speedups of the entire computation as we use more computing nodes. A more explicit solution is to overlap the *Places* initialization and computation in a pipelined fashion, as MASS currently facilitates pipelined graph streaming which partially performs graph computing while constructing a next portion of a given graph [8].

B. Data Retrieval

As shown in Figure 7, the retrieval performance of MASS library depends on the size of an input matrix A. MASS cannot outperform the sequential execution when sorting and retrieving top data items from a small size of matrix such as

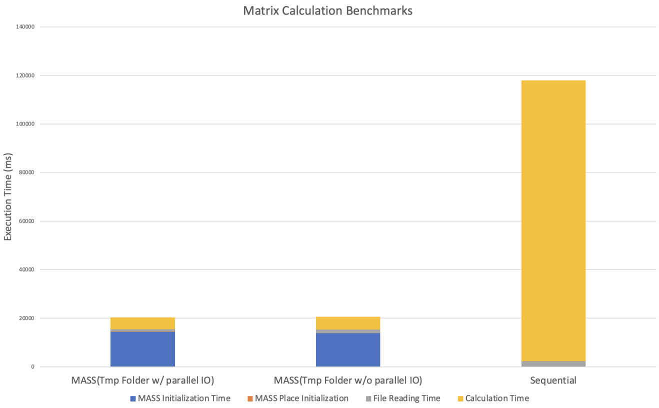


Fig. 6. Matrix multiplication benchmarks

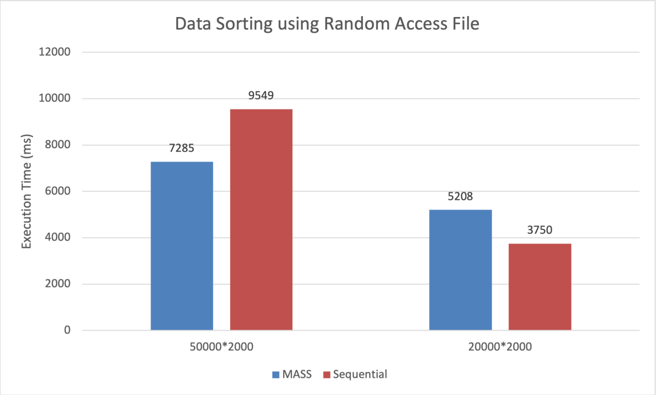


Fig. 7. Data sorting with memory space

20,000 data items ×2,000 features. It runs 38.8% slower than the sequential execution. However, if the matrix size gets 2.5 times larger, (i.e., increased to 50,000 items while keeping the same number of features, (i.e., 2,000), MASS agents can keep their propagation overheads within 1.4-times slow-down, which is from 5,208 milliseconds to 7,285 milliseconds. On the other hand, the sequential execution takes 9,549 milliseconds as compared to 3,750 milliseconds in the smaller matrix, thus resulting in a 2.5-time, linear increase of overheads.

This is because MASS agent propagation can identify a smaller number of top candidate data items, (e.g., 3,831 in the larger matrix) while the sequential execution needs to exhaustively sort all 50,000 data items.

V. Related Work

This section summarizes some notable parallel database systems, semantic database examples, and agent-based ensemble of multiple database systems. Thereafter, we differentiate our agent-based MMM parallelization from these related systems.

A. *Parallel Database*

There are three types of architecture for parallel databases: shared-disk, shared-memory, and shared-nothing architectures [9]. Shared-memory architecture facilitates a global memory to all computing nodes. In [10], authors propose multiple algorithms that use remote direct memory access (RDMA) in parallel database systems. The work dynamically manages RDMA-registered memory to improve the database performance. A shared-disk parallel database system is proposed in [11]. Shared-disk systems manage to lower communication overheads as compared to other architectures. Especially for reading operations, the systems spread the operations over multiple nodes to reduce the response time.

Our MMM and MASS integrated solution is a shared-nothing architecture. The advantage of shared-nothing architecture realizes high extensibility and high availability. Compared with other shared-nothing architecture, such as MySQL Cluster[4], MASS is easier to scale up the system without manually partitioning data over a parallel system. Our approach is also differentiated from the viewpoint of initiating agent propagation from a test data rather than screening all potential data items. Since each agent injection represents a different database query, multiple transactions can be carried out in parallel.

B. *Semantic Database*

Semantic Data Model (SDM), introduced by Michael Hammer, provides a high-level semantic-based modeling mechanism to capture and express the structural formalism for databases [12]. SDM facilitates data querying from different perspectives, where users can query data items by declaring their views of a large database. Although SDM leads to some redundancy of data storage by providing multiple perspectives of a database, users can still benefit from its enriched relationship schema to better understand the data in a natural way.

In World Wide Web, W3C introduced the standard of Web Ontology Language (OWL) and the Resource Description Format (RDF) to realize a semantic web. The purpose of introducing a semantic web is to have machines understand and interpret complex human requests based on their meaning[5]. OWL and RDF use a triplet of (subject, predicate, object). The subject and object denote resources. The predicate tells subject-object relationship and describes their traits or aspects. OWL and RDF utilize a meta-data model to express semantic meaning in web resources and various syntax notations to make the semantic meaning understandable by machines. With a growing amount of web information being processed and extracted, the most useful and related information can be filtered by domains. Also, web crawlers can be beneficial from the semantic web information. In [13], a semantic-based model to represent multimedia big data is proposed. The work describes a property-based graph which allows users to express the concept and relationship between multimedia data items. A graph database is utilized to save data items in a key-value pair format and to traverse their connections. Their work presents a machine-understandable representation which organizes the semantic associations between multimedia data items.

[4] https://www.mysql.com/products/cluster/

[5] https://www.w3.org/standards/semanticweb/

Most semantic database models highly rely on either traditional models in the relational database architecture or on models of resources and their relationship [13]. In addition, some semantic databases are limited to specific domains where they usually do not provide a general semantic database solution. Contrary to them, MMM does not rely on any relational database. A group of features represents the value of data items. In addition, MMM can realize the dynamic projection to a semantic space from user-given context words.

C. Agent-based Database

Most uses of agents in database or data-storing systems focus on orchestrating multiple databases or backup storages but not on parallel data retrieval within each computing node. Agent-based backup [14] monitors and maintains the data consistency among multiple storages. Archiware is a commercial system to facilitate secured, recoverable, and distributed data backup without using a centralized manager[6].

Agent-based data mining [15] dispatches an agent to each of remote and different databases where it performs mining algorithms best fit to the respective dataset. The mined results are collected by an aggregation agent and are presented to the user. Similarly, agent-based data reduction carries out multi-database mining but also conducts reductive computation with agents before presenting the final results back to users [16].

Our parallelization approach is different from these agent-based databases in the viewpoint of how to involve agents into data retrieval or mining. Contrary to the related systems that use agents for inter-database monitoring, task coordination, and/or reductive computation, our agent-based MMM parallelization injects agents into a semantic space to retrieve data through agent propagation. Therefore, our approach can even parallelize data retrieval within a single database system.

VI. Conclusion

We used two agent-based parallelization techniques to scale up and to accelerate an MMM semantic database. One is agent-based matrix computation. Upon any modifications of data items (including data insertions, deletions, and feature updates), MMM needs a significant amount of mathematical computation to construct a correlation matrix and compute its eigenvectors. It takes over 40 minutes for an input matrix of 40,000 data items × 2,000 features. We applied agents to Cannon's algorithm to speed up matrix multiplication 22 times faster. The other technique is agent propagation over a semantic space to find data items closer to a test item, which is quicker than exhaustive data sorting by the sequential program. Although the performance improvement with agent propagation is only 23.7%, we expect that this would support larger datasets more efficiently.

While we used MASS for parallelizing a multi-dimensional semantic database, any other multi-agent systems such as RepastHPC [17] are applicable to our parallelization work as far as they can dispatch agents over a logical space that is mapped to a cluster system.

Finally, our future work includes the following two tasks: (1) pursuing further speed-up with agent-based parallelization of eigenvector decomposition and (2) measuring the maximum query-handling throughput of MASS-based MMM.

Acknowledgments

We would like to extend our deep appreciation to Prof. Yasushi Kiyoki and his research group at Keio University at Shonan Fujisawa Campus, Japan for all their technical advice on our Java implementation of MMM algorithms as well as their support for our verification work.

References

[1] T. Kitagawa and Y. Kiyoki, "A mathematical model of meaning and its application to multidatabase systems," in *RIDE-MS '03: 3rd International Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems*, Vienna, Austria, April 1993, pp. 130–135.

[2] M. Fukuda, C. Gordon, U. Mert, and M. Sell, "Agent-Based Computational Framework for Distributed Analysis," *IEEE Computer*, vol. 53, no. 3, pp. 16–25, 2020.

[3] Y. Kiyoki, T. Kitagawa, and T. Hayama, "A metadatabase system for semantic image search by a mathematical model of meaning," *ACM SIGMOD RECORD*, vol. 23, pp. 34–41, December 1994.

[4] H.-J. Lee, J. P. Robertson, and J. A. Fortes, "Generalized cannon's algorithm for parallel matrix multiplication," in *11th international conference on Supercomputing*, New York, NY, July 1997, pp. 44–51.

[5] Y. Kiyoki and X. Chen, "A semantic associative computation method for automatic decorative-multimedia creation with "kansei" information," in *6th Asia-Pacific Conference on Conceptual Modeling (APCCM 2009)*, Wellington, New Zealand, January 2009, pp. 7–15.

[6] Random House, *Roget's Thesaurus*. New York: Ballantine Books, 2001.

[7] F. Rotella and I. Zambettakis, "Block householder transformation for parallel qr factorization," *Applied Mathematics Letters*, vol. 12, pp. 29–34, May 1999.

[8] Y. Hong and M. Fukuda, "Pipelining Graph Construction and Agent-based Computation over Distributed Memory," in *9th Int'l Workshop on BigGraphs'22*. IEEE, December 2022, pp. 4616–4624.

[9] A. S. Talwadker, "Survey of performance issues in parallel database systems," *Journal of Computing Sciences in Colleges*, vol. 18, pp. 5–9, June 2003.

[10] F. Liu, L. Yin, and S. Blanas, "Design and evaluation of an rdma-aware data shuffling operator for parallel database systems," *ACM Transactions on Database Systems*, vol. 44, no. 4, pp. 1–45, December 2019.

[11] E. Rahm, "Parallel query processing in shared disk database systems," *ACM SIGMOD RECORD*, vol. 22, pp. 32–37, December 1993.

[12] M. Hammer and D. McLeod, "The semantic data model: a modelling mechanism for data base applications," in *ACM SIGMOD International Conference on Management of Data - SIGMOD '78*, Austin, TX, 1978, pp. 26–36.

[13] A. M. Rinaldi and C. Russo, "A semantic-based model to represent multimedia big data," in *10th International Conference on Management of Digital EcoSystems*, Tokyo Japan, September 2018, p. 31–38.

[14] H.-M. Lee and C.-H. Yang, "A distributed backup agent based on grid computing architecture," in *9th international conference on Knowledge-Based Intelligent Information and Engineering Systems - Volume Part II*, September 2005, pp. 1252–1257.

[15] P. Yang, L. Tao, L. Xu, and Z. Zhang, "Multiagent framework for bio-data mining," in *International Conference on Rough Sets and Knowledge Technology*, Gold Coast, Australia, July 2009, pp. 200–2007.

[16] I. Czarnowski and P. Jedrzejowicz, "Instance reduction approach to machine learning and multi-database mining," *Annales UMCS Informatica AI*, vol. 4, no. 1, pp. 60–71, 2006.

[17] RepastHPC, "Accessed on: June 8, 2023. [Online]. Available: https://repast.github.io/repast_hpc.html."

[6] https://www.archiware.com/