2014 IEEE/ACM International Symposium on Big Data Computing

Progressive Data Stream Mining and Transaction Classification for Workload-Aware Incremental Database Repartitioning

Joarder Mohammad Mustafa Kamal
Faculty of Information Technology
Monash University
Victoria, Australia
Email: joarder.kamal@monash.edu

Manzur Murshed
Faculty of Science and Technology
Federation University
Victoria, Australia
Email: manzur.murshed@federation.edu.au

Mohamed Medhat Gaber
School of Comp. Science and Digital Media
Robert Gordon University
Aberdeen, United Kingdom
Email: m.gaber@rgu.ac.uk

***Abstract*—Minimising the impact of distributed transactions (DTs) in a shared-nothing distributed database is extremely challenging for transactional workloads. With dynamic workload nature and rapid growth in data volume the underlying database requires incremental repartitioning to maintain acceptable level of DTs and data load balance with minimum physical data migrations. In a workload-aware repartitioning scheme transactional workload is modelled as graph or hypergraph, and subsequently perform k-way min-cut clustering guaranteeing minimum edge cuts can reduce the impact of DTs significantly by mapping the workload clusters into logical database partitions. However, without exploring the inherent workload characteristics, the overall processing and computing times for large-scale workload networks increase in polynomial orders. In this paper, a workload-aware incremental database repartitioning technique is proposed, which effectively exploits proactive transaction classification and workload stream mining techniques.Workload batches are modelled in graph, hypergraph, and compressed hypergraph then repartitioned to produce a fresh tuple-to-partition data migration plan for every incremental cycle. Experimental studies in a simulated TPC-C environment demonstrate that the proposed model can be effectively adopted in managing rapid data growth and dynamic workloads, thus progressively reduce the overall processing time required to operate over the workload networks.**

***Keywords*-Cloud databases; workload; distributed transactions; data stream mining; classification; incremental repartitioning; load-balance; data migration;**

I. Introduction

Rapid growth in 'Big Data' dimensioning at dynamic velocity, volume, and variability challenges real-time processing and computation in the Cloud. Modern user-facing scalable Cloud applications driving Online Transaction Processing (OLTP) workloads require *best-effort* throughput and latency guarantees from the backend shared-nothing database services. However, the underlying system fails to guarantee such scalability requirements at the front end Web tiers without scaling-out at the back end databases [1]. At the same time, maintaining ACID (Atomicity, Consistency, Isolation and Durability) properties within a relational database cluster while ensuring high-scalability requirements is a difficult trial for the system owners. Due to skewed data popularity and OLTP dynamics, DTs frequently appear in the databases and usually require distributed consensus protocols like '2-Phase Commit (2PC)' [2] to execute across the shared-nothing cluster. However, the consequence of using 2PC in a scalable OLTP Cloud database serving billions of online users can be severe [3]. More adversely, this leads to I/O overhead, resource contentions, increase latency, distributed joins, and deadlocks in geo-distributed Cloud databases [4].

For many years, logically partition a shared-nothing database based on the initial assumptions of workload characteristics plays an important role in setting up a robust backend OLTP database service. However, this static solution fails to adopt with continuous workload changes, and requires extensive administrative interventions. In recent years, workload-aware static [5] and incremental repartitioning [6] methods are proposed where transactional workloads are represented as graph or hypergraph. In these workload representations, an edge links two tuples (i.e., vertices) involved in the same transaction on a graph, while a hyperedge connects all of its tuples on a hypergraph respectively. To reduce the severity of DTs, these graphs are then clustered using k-way min-cut [7] ensuring minimum transactional edge cuts, thus group the tuples from the same transactions into the same cluster.

Later, these workload clusters are mapped into logical database partitions initially created. The tuple-to-partition mappings are typically stored centrally in a router to serve the transactional queries. Incremental repartitioning involves intra- and inter-server physical data tuple migrations at lean operational periods to maintain the initial min-cut gains based on a predefined threshold – percentage of DTs within a workload batch. However, this reactive approach can be ineffective at handling situations like sudden Web traffic spike or, serving a large number of database requests after a major software update. At the same time, finding potential candidate tuple sets for data migrations which can certainly reduce the impact of DTs under a fixed threshold is often an inconvenient task to manage. Finally, there is hardly any check and balance between server-level load-balance and the amount of inter-server data migrations to be carried out.

In this paper, we propose a proactive incremental database repartitioning technique which effectively exploit progressive data stream mining in identifying frequently accessed tuple sets from the workload and classify the transactions accordingly. We freshly create workload representative networks at fixed interval periods i.e., hourly or daily, and use the graph min-cut algorithm to produce fresh clusters to be randomly one-to-one mapped into the logical database partitions. To reduce the processing time of incremental min-cut process (which is a known *NP*-hard problem) and reduce the size of the workload networks we continuously mine transactional log streams and classify the transactions to identify the most frequently occurred DTs and non-DTs whose tuples can be selected as potential data migration candidates.

Furthermore, we explored the possibility of producing fixed and dynamic number of clusters from the min-cut process. The number of clusters can be the same as the number of logical partitions–*fixed*, or varied in accordance with the selection of partitions which are only involved in the current workload network–*dynamic*. Rather than using a central lookup router we effectively use the concept of *roaming* for tuple lookup to ensure high-scalability in transaction processing. We evaluated our proposed technique using different workload representations–fine, exact, and coarse granularity as graph (GR), hypergraph (HGR), and compressed hypergraph (CHG) to understand the suitability in creating workload networks. We measure the dimensionality of challenges–the impact of DTs, load-balance, and data migrations–involved in incremental repartitioning.

Three distinct repartitioning performance metrics are used to observe the fulfillment of the intended goals in a simulated environment using TPC-C workload-sensitive batches for 100 incremental periods. In preference of observing the effect of progressive data stream mining and transaction classification for intelligent data analysis, we compare the experimental results using three distinct classification settings namely–*BC*, *FD*, and *FDFND* against the baseline situation where no classification *NC* is used. Our findings show that proactive stream mining and classification in workload analysis is effective in reducing the processing time of database repartitioning and suitable for adopting in very large-scale Big Data computing.

The remainder of this paper is organised as follows: related works are discussed in Section II; Section III details the proposed incremental repartitioning technique with notations and a high-level system overview with related steps; Section IV discusses the experimental results and analysis; and finally Section V concludes the paper with future work directions.

II. Related Works

Distributed query processing has been a well discussed topic in computing research, and significant efforts have been spent to establish its background, theories, challenges, and practical aspects for real-life implementations [8]. With increase diversification of Big Data applications, workload-aware incremental database repartitioning has gained significant momentum over the past few years. [5] is one of the first to propose *Schism* which represents transactional workload as graph and then performs k-way replicated clustering to minimise the effect of distributed transactions. However, large-scale graphs are required to generate in Schism as it explodes each tuple as a star-shaped way with its transactional frequency. Similar approaches are also proposed at [6], [9], and [10] where hypergraphs and compressed hypergraphs are used to represent transactions in a workload network for both transactional and analytical workloads. Both of the techniques consider dynamic workload changes and propose to move fixed amount of data in a regular interval with a predefined threshold parameter while finding the dense subgraphs of a specified size. [11] propose CloudTPS which considers short-lived transactions, identifications of data items that are frequently accessed by the transactions, and inconsistent data retrieval of read-only transactions. However, CloudTPS neither consider incremental repartitioning nor real-time analysis of transactional log streams. More recently, [12] has proposed a temporal activity hyper-graph for social networks to minimise the number of distributed transactions using k-way balanced clustering with minimum I/O overheads and server imbalance. None of the works mentioned above explore intelligent data analysis techniques on database logs generated by periodical workloads. [13] propose LuTe which maintains a router-based lookup table for partitioned data sets which produces faster lookups for scalable transaction processing but it did not consider frequently accessed transactional patterns and the necessity of incremental repartitioning.

We argue that existing models for mining transactional logs can be effectively used to identify the *hot* and most frequently accessed tuples in the workload. In [14] the authors proposed a techniques to mine time-sensitive transactional patterns with approximate support over frequent and semi-frequent. [15] proposed the notion of hierarchical heavy hitters and showed effective way to keep track of the hottest tuples in a less computational way. Later, [16] proposed semi Frequent Closed Itemsets (semi-FCIs) in their *IncMine* algorithm to incrementally track the tuples that might become frequent and hot over the period of time. [17] focuses on mining frequent items from distributed data stream sources and can be effectively used in workload data analysis for distributed shared-nothing databases. An active learning approach is proposed in [18] which uses a pre-clustering process to select most informative instances from transaction sensitive workload batches for training the learner. Interested readers can find a detail review of techniques for finding frequent items over data streams at [19].

III. Incremental Workload-Aware Repartitioning

A. System Overview

We consider the architecture of a typical 3-tier Web application which is commonly used from financial to social network processing for discussing the proposed incremental repartitioning technique as shown in Figure 1. Based on common deployment practice, a 3-tier application framework in the Cloud is composed of a series frontend Web servers following by a group of Application servers balancing the workloads, and finally the backend Database servers serve

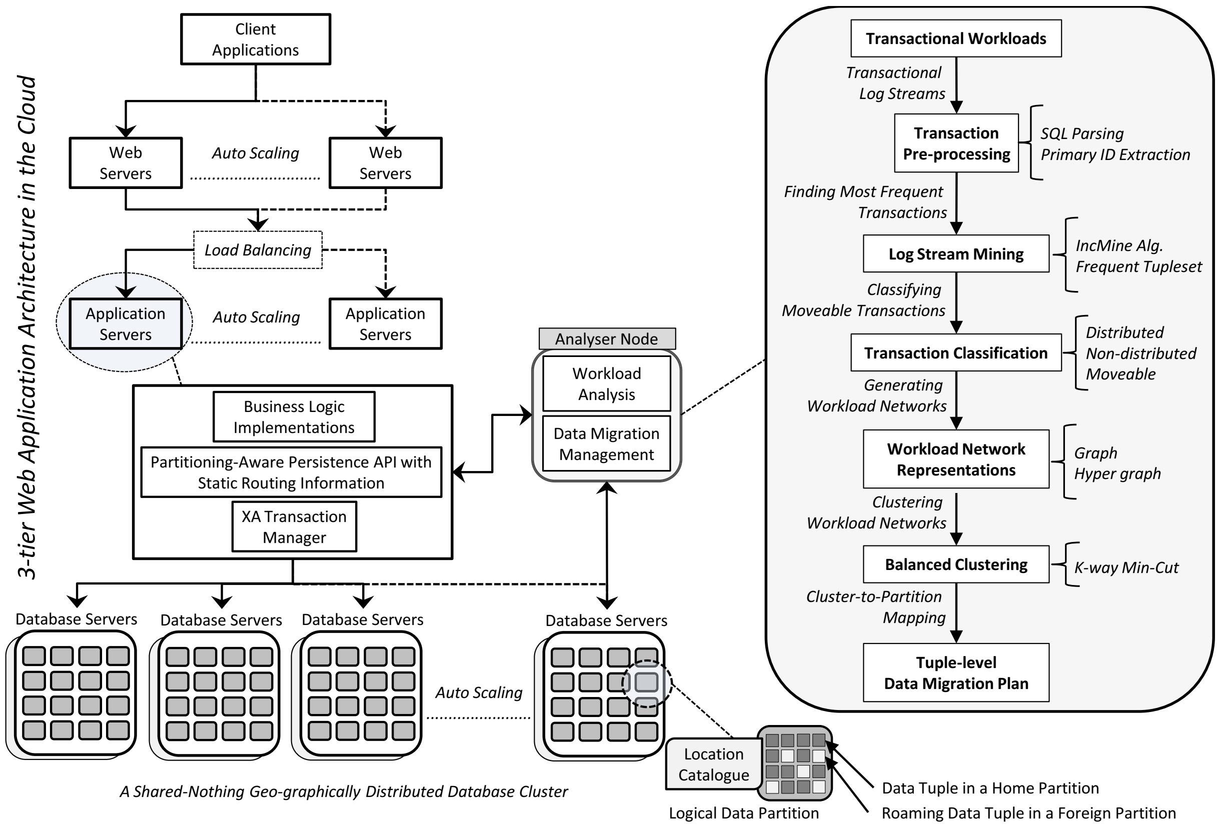
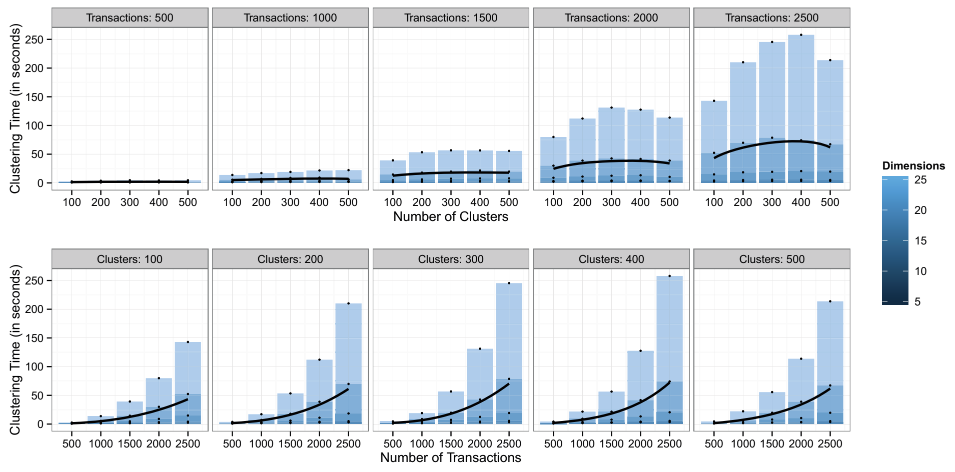

3-tier Web Application Architecture in the Cloud
Client Applications
Web Servers
Auto Scaling
Web Servers
Load Balancing
Application Servers
Auto Scaling
Application Servers
Business Logic Implementations
Partitioning-Aware Persistence API with Static Routing Information
XA Transaction Manager
Analyser Node
Workload Analysis
Data Migration Management
Database Servers
Database Servers
Database Servers
Auto Scaling
Database Servers
A Shared-Nothing Geo-graphically Distributed Database Cluster
Location Catalogue
Logical Data Partition
Data Tuple in a Home Partition
Roaming Data Tuple in a Foreign Partition
Transactional Workloads
Transactional Log Streams
Transaction Pre-processing
SQL Parsing
Primary ID Extraction
Finding Most Frequent Transactions
Log Stream Mining
IncMine Alg.
Frequent Tupleset
Classifying Moveable Transactions
Transaction Classification
Distributed
Non-distributed
Moveable
Generating Workload Networks
Workload Network Representations
Graph
Hyper graph
Clustering Workload Networks
Balanced Clustering
K-way Min-Cut
Cluster-to-Partition Mapping
Tuple-level Data Migration Plan

Fig. 1. An overview of the workload-aware incremental repartitioning framework representing the overall work flows. Steps inside the shaded rounded rectangle represent the flows of workload analysis, representation, clustering, and repartitioning decision generation.

the transactional queries while setup within geo-distributed data centers. Focusing on the particular components of an Application server reveals the implementation of business logics, maintenance of static and dynamic routing information for logical database partitions, and coordinating of DTs using XA [20] transaction manager. We propose the incorporation of an *Analyser* node working in parallel with the Application servers which continuously process the transactional workload streams to identify the most frequent DTs containing the most frequently accessed tuples from the underlying database. As shown in the figure, a Transaction Classification process is followed by the Log Stream Mining which identifies the moveable DTs and non-DTs (later discussed in Section 4). Finally, balanced min-cut clustering is carried out to produce k fixed and dynamic number of clusters with a tuple-level data migration plan to be carried out. The analyser node is also responsible to carry of the tasks of intra- and inter-server physical data migrations without interrupting the processing of ongoing transactions in the Application servers by utilising the roaming capability of our proposed system. A logical data partitions is also highlighted to show the data tuples residing in their home and foreign partitions having a roaming catalogue. The details of each of the steps shown inside the shaded rounded rectangular box are discussed in following.

B. The Cost of k-way Min-Cut Clustering

In a workload-aware database repartitioning approach, the size of the workload network represented as graph, hypergraph, or even in their compressed forms plays a significant role in clustering performance and consequently in the overall repartitioning performance. From our three-dimensional empirical analysis we found that clustering times increase in polynomial order with the size of workload networks and transactional dimensions. We present the observation of the growth of computing time for performing k-way balanced clustering of workload networks represented by transactional hypergraphs in this particular case. Three observational dimensions are chosen to understand the consequences. At one extreme, we extend the transactional dimensions (i.e., the number of tuples within a transaction) to produce simple to more complex transactional signatures. At another extreme, we simultaneously increase the number of transactions as well as the number of target clusters defined by k. Figure 2 shows the growth of computing time for simultaneous k-way clustering of transactional hypergraphs with increasing number of clusters (in top) and with increasing number of transactions (in below). We able to fit fourth order polynomial



(a)

Fig. 2. Clustering time for workload networks represented as hypergraph with fourth order polynomial fitting.

curves in these plots which clearly show the adverse impact on clustering time having large workload networks or targeted number of clusters with varying transactional dimensions.

These observations justify our envision in mining transactional log streams and classify incoming user queries. This intuition lead us in searching the most frequently occurred transactions and those which are intertwined with them. Later, only these selective set of transactions are used to represent the workload networks which are smaller in size comparing to use every transactions present in a particular workload batch. In contrast, reducing the size of workload network and targeted clusters may influence the impact of distributed transactions and may also lead to data distribution imbalance. Yet, the achieved benefits of minimising physical data movements along with faster workload network processing certainly overpower the drawbacks, thus, increase the overall repartitioning performance as will be shown in later sections.

C. Mining Distributed Transactions Containing Semi-Frequent Closed Tuple Sets

Let $S = \{S_1, S_2, ..., S_N\}$ be the physical servers supporting a OLTP database and consisted of $P = \{P_1, P_2, ..., P_M\}$ logical table partitions evenly distributed among them. Furthermore, $\mathcal{D}_{S_i} = \sum_{\forall j} \mathcal{D}_{P_{ij}}$ be the amount of data volume in S_i consisting of a set of logical partitions $\forall j P_{ij}$. Let, $\mathcal{W} = \{\mathcal{W}_1, \mathcal{W}_2, ..., \mathcal{W}_n\}$ represents a *transaction-sensitive* set of workload batches containing τ transactions each and $\mathcal{W}_i$ be the workload at ith round. A typical $\mathcal{W}_i$ contains queries submitted by the end users from the predefined set of transactions $T = \{T_1, T_2, ..., T_z\}$ where any transaction T_i can appear multiple times within W_i having a frequency of $freq(T_i)$. Furthermore, temporal weights $temp(T)$ are used to prefer the preceding transactions over the succeeding ones.

For any given $\mathcal{W}_i$ and σ the *minimum support threshold* (i.e., *MST*), our aim is to identify the set of all *frequent closed tuple sets* over the stream where the *tuple sets* are *distributed* i.e., not originated from the same physical server. Let $\mathcal{D} = \{d_1, ..., d_t\}$ be the set of all tuples within the database thus any *transactional tuple set* (t_{ts}) will be a proper subset of $\mathcal{D}$. Again let the support of any t_{ts} over $\mathcal{W}_i$ be $sup(t_{ts_i}, \mathcal{W}_i)$ representing the number of transactions in $\mathcal{W}_i$ that support t_{ts_i}. Following the notion of *semi-frequent closed tuple sets*(*semi-FCTS*) [16], let ϵ be the *maximum support error* or, *relaxed MST* and r be the *relaxation rate* to find the *semi-FCTS* over each workload batch $\mathcal{W}_i$. Let, $minsup(i)$ be the incremental *MST* function for $\mathcal{W}_i$ which is defined in (1) in below.

$$minsup(i) = \lceil (m_i r_i) \rceil; \tag{1}$$

where *min_support*, $m_i = \sigma\tau$ and $r_i = (\frac{1-r}{\tau})(i-1) + r$ which gradually increases the value of r for each succeeding workload batches. Any t_{ts_i} is kept if and only if its approximate support over most recent workload batches $\widetilde{sup}(t_{ts_i}, \mathcal{W}_i)$ is no less than $minsup(i)$. Any T_i will contain a *semi-FCTS* within $\mathcal{W}_i$ if and only if both (2) and (3) hold true.

$$\exists i \mid \widetilde{sup}(t_{ts}, \mathcal{W}_i) \geq minsup(i) \tag{2}$$

$$\nexists t_{ts_j} \supset t_{ts_i} \mid \widetilde{sup}(t_{ts_j}, \mathcal{W}_i) = \widetilde{sup}(t_{ts_i}, \mathcal{W}_i) \tag{3}$$

D. Proactive Transaction Classification

Typical transaction level workload sampling process identifies most frequently submitted database transactions by the end users and separates the distributed ones. In course of our experimentations we found that there exist a group of tuple sets which are frequent and appear in both distributed and non-distributed transactions. We argue that due to the presence of these tuples in distributed transactions while performing k min-cut clustering followed by physical data migrations the previously non-distributed transactions became distributed,

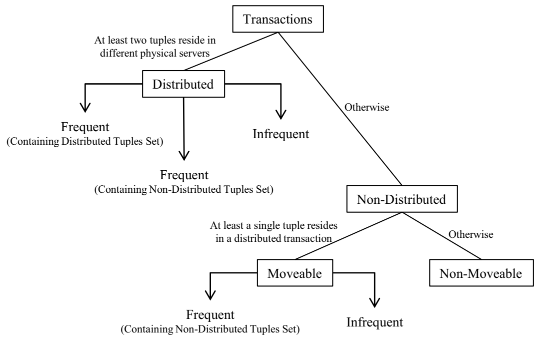

Transactions
At least two tuples reside in different physical servers
Distributed
Otherwise
Frequent (Containing Distributed Tuples Set)
Infrequent
Frequent (Containing Non-Distributed Tuples Set)
Non-Distributed
At least a single tuple resides in a distributed transaction
Otherwise
Moveable
Non-Moveable
Frequent (Containing Non-Distributed Tuples Set)
Infrequent

Fig. 3. Proactive transaction classification tree.

and thus increase the computing and processing time of any workload-aware incremental repartitioning scheme.

Based on the above heuristic, a proactive classification tree has been proposed as shown in Figure 3. Transactions are classified as *Distributed* containing at least two tuples residing in different physical servers. Otherwise, *Non-Distributed* transactions are further classified as *Moveable* and *Non-Movable*. While the former type should contain at least a single tuple which also present in any of the *Distributed* transaction, the later type only represents transactions containing tuple sets that are purely non-distributed and hereby out of the interest. By utilising progressive data streaming mining described above both of the *Distributed* and *Movable Non-Distributed* transactions are further classified in the basis of containing Semi-FCTS otherwise the they are tagged as *Infrequent*. Furthermore, the selective transactions are classified into groups based on whether the contained *semi-FCTS* are themselves distributed or not. Effectively this proactive classification reduces the sampled workload size hence the size of representing graphs or hypergraphs, and thereby their processing and computing time as well.

From the above classification we set three different classification techniques–1) Base Classification (*BC*) (containing moveable frequent and infrequent distributed or non-distributed transactions having both distributed and non-distributed tuples); 2) Frequent Distributed (*FD*) (containing moveable frequent distributed transactions having only distributed tuples); and 3) Frequent Distributed and Frequent Non-Distributed (*FDFND*) (containing moveable frequent distributed transactions having both distributed and non-distributed tuples).

E. Workload Representation

Any W_i can be represented by a graph $\mathcal{G} = (\mathcal{V}, E)$ where an edge $e = \{v, u\}$ with a co-access frequency weighted by w_e connects a pair of vertices from V and each vertex having a weight w_v depicting its frequency. A vertex within the graph represents a *data tuple* while an each edge connects the tuples from the set of adjacent vertices for a given $v \in V$ and emerged from the same transaction.

A hypergraph, $\mathcal{H} = (\mathcal{V}, E)$ on the other hand represents the workload network where a hyperedge $e \in E$ having its frequency weighted as w_e represents a transaction and $V_e \subseteq \mathcal{V}$ represents the corresponding tuples. $\mathcal{H}$ can be further compressed to $\mathcal{H}_c = (\mathcal{V}', E')$ representing a compressed hypergraph where each $e' \in E'$ is the set of virtual vertices $v'_{e'} \subseteq \mathcal{V}'$ and the original vertices of e can be mapped into $|v'_{e'}| \geq 2$. C_r denotes the level of compression adopted (i.e., compression ratio) and equals to $|V|$ for full compression and to 1 for none.

F. k-way Balanced Clustering and Repartitioning

Incremental database repartitioning is accomplished by performing k-way clustering of the classified workload representation in such a way that minimum transactional edges are cut. By keeping the clustered tuple sets together in a same physical server most of the distributed transactions will convert to non-distributed ones. For the non-distributed transactions included in the k-way clustering from the classification step, with a high probability they will remain non-distributed to ensure minimum edge cut within the clustering process. The *balance* criteria of the k-way clustering process ensures that *imbalance ratio* of $k \max(w_{v_x})/W(\mathcal{V})$ equals or close to 1 where $W(\mathcal{V}) = \sum_{\forall x} w_v$.

Given the representation of the classified workload, $\mathcal{G}$ and a maximum allowed imbalance ratio ε, the k-way clustering $\Pi^{\mathcal{G}} = \{V_1, V_2, ..., V_k\}$ will minimise the transactional edge-cuts having a *balance* constraint bounded by $(1+\varepsilon)$. Similarly, for the hypergraph representation $\mathcal{H}$ the k-way clustering will be $\Pi^{\mathcal{H}} = \{\mathcal{V}_1, \mathcal{V}_2, ..., \mathcal{V}_k\}$ ensuring the constraint impose by ε such that the minimum number of hyper-edges are cut. The k-way clustering of $\mathcal{H}_c$ will be $\Pi^{\mathcal{H}_c} = \{\mathcal{V}'_1, \mathcal{V}'_2, ..., \mathcal{V}'_k\}$ with the *balance* constraint aiming at the minimum edge cuts.

Finally, the clustered tuple sets need to be mapped in the logical database partitions in such a way so that tuples from the same distributed transactions reside together. We choose to perform the k-way clustering at the partition level rather than in physical server level which eventually shuffles the tuples in a more fine granularity and has been confirmed from our initial experimentations. It also provides greater degree of freedom to ensure both server and partition level tuple balance. A *random one-to-one* cluster-to-partition mapping strategy is used to move the tuple sets which inherently ensures tuple balance within the partitions as well as in the servers.

With the aim of reducing the size of given workload representation, we argue that if the value of k is chosen *dynamically* based on the number of partitions representing the transactions in the sampled workload the overall generation, processing and computing time of individual workload representations will reduce significantly in compare to use *fixed* number of partitions. Our experimental findings also support this argument hence show its effectiveness as shown in later sections. In preference of discussing the effects of proactive transactional log stream mining and classification, we do not discuss the process of distributed tuple lookup using the concept of *roaming* [21] in detail.

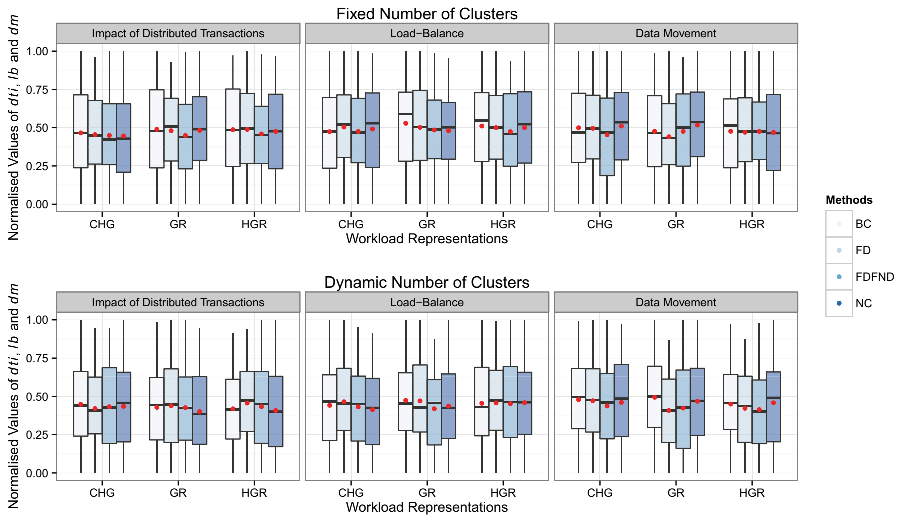


Fig. 4. Comparison of different methods with fixed and dynamic number of clusters.

G. Performance Measure Criteria

1) Minimise The Impact of Distributed Transaction: Let $costs(T_i)$ be the cost of spanning the physical servers by any T_i and can be calculated as $|S_{T_i}| - 1$ where S_{T_i} is the number of physical servers spanned by T_i. By multiplying $cost_s(T_i)$ with $freq(T_i)$ and $temp(T_i)$ the mean impact of the distributed transactions T^d can be determined for W_i as show in (4).

$$dti = \left(\sum_{\forall T_i \in T} cost_s(T_i) freq(T_i) temp(T_i) \right) / |T^d| \quad (4)$$

2) Minimise Load Imbalance: We measure the coefficient of variation (CV) for all the servers under the consideration which tells the variability of tuple distribution within the servers with respect to the mean data volume $\mu_{\mathcal{D}_S}$. (5) in below determines CV of the load balance measure lb.

$$lb = \sigma_{\mathcal{D}_S} / \mu_{\mathcal{D}_S} \quad (5)$$

where $\mu_{\mathcal{D}_S} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{D}_{S_i}$ and $\sigma_{\mathcal{D}_S} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (\mathcal{D}_{S_i} - \mu_{\mathcal{D}_S})^2}$

3) Minimise Inter-Server Data Migrations: The effect of incremental repartitioning is physical data movements between the logical partitions and physical servers. And we only prefer the *inter-server* data movements that potentially restrict database throughput and increase latency. For a given W_i the count of *inter-server* data migration can be normalised by dividing with the mean data volume $\mu_{\mathcal{D}_S}$ to evaluate the data migration measure dm as shown in (6).

$$dm = MV / \mu_{\mathcal{D}_S} \quad (6)$$

where MV is the total number of the tuples interchanging physical servers.

4) Minimise Processing and Computing Time: We perform a five-way measurement of processing time which quantifies the time required to perform data stream mining, base classification (without performing steam mining), generate respective workload representation, k-way clustering and finally physical data movements. Furthermore, in case of using *dynamic* k values against *fixed* ones processing and computing times are also measured to understand their individual effect.

IV. Experimental Results

We develop a workload-driven simulator to analyse the effectiveness of the proposed scheme with synthetically generated TPC-C workload with 10 Warehouses. The targeted OLTP database is distributed over 10 homogeneous physical servers consisting of 90 logical table partitions in total. The workload is modelled following [22] and [23] to dynamically vary based on data popularity following Zipf's law and with consistent data volume growth for 100 workload batches. Workload networks are represented by graphs *GR*, hypergraphs *HGR* and compressed hypergraphs *CHG* (with C_r = 0.5). We incorporated the implementation of *IncMine* [24] in our simulation model, and extend it to mine the distributed and non-distributed *semi-FCTS* to support the proposed proactive transaction classification process.

We use the incremental performance measures described in (4), (5), and (6) for evaluating three different setups using GR, HGR, and CHG workload representations. The evaluated performance metrics (dti, lb, and dm) are normalised using *0-1 Normalisation* and presented for both *fixed* and *dynamic* number of targeted clusters. We compare the proposed transaction classification technique in three different settings with the baseline of *No Classification (NC)* for both fixed and dynamic values of k as shown in Figure 4 with bar plots and mean points in red dots.

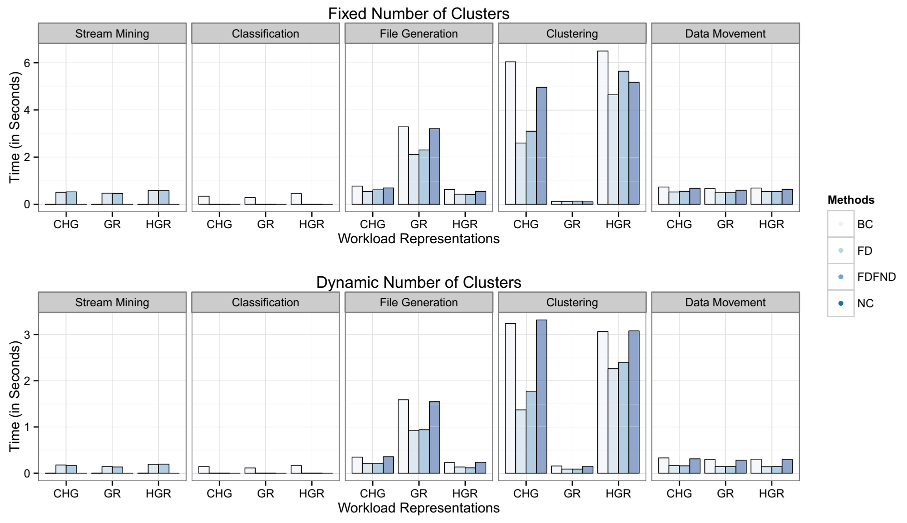


Fig. 5. Processing and computing times with fixed and dynamic number of clusters.

Among the variations the first method comprises the *BC* hierarchy which includes all distributed and movable non-distributed transactions into consideration. The next method *FD* restricts the selection of transactions to only the ones which are frequent and contain distributed semi-FCTS. Finally, the *FDFND* method that selects the transactions that are distributed or movable and contain both distributed or non-distributed semi-FCTS. The processing times for these three distinctive classification setup is shown in Figure 5 for fixed and dynamic number of clusters.

A. Analysis of dti, lb, and dm

From the top plot in Figure 4, *FDFND* based methods with *HGR* representation perform better than all others in most of the cases. Similarly *FD* methods with both *GR* and *HGR* representations performed well. With respect to performance criteria, *HGR-FDFND* reduces the impact of distributed transactions to great extent and with optimum physical data movements. On the other hand, both *GR-FD* and *GR-FDFND* handles the load balance criteria in a much better way comparing to others. Now, similar observations can be also found with dynamic number of clusters but at a lower intensity (i.e., all of the values are less than 0.5) as shown in the bottom plot of Figure 4.

This observation clearly supports our claim of dynamically vary the number of targeted clusters. On the contrary, potential load imbalance can occur while dynamically varying the targeted k value in the long run within the lifetime of a distributed database and different heuristic or meta-heuristic based optimisation techniques can be used for remedy. However, at this moment we are primarily focusing on providing the best effort solutions and vary the underlying workload representation and classification methods to handle the situations accordingly.

B. Analysis of Processing and Computing Times

As mentioned earlier, processing and computing times are observed for five different activities as shown in Figure 5. Note that, for *BC* based methods there are no timing information for *Stream Mining* and similarly for *NC* methods no bars represent the timing for both *Classification* and *Stream Mining*. For the other three activities namely workload file generation, clustering, and physical data migrations, *FDFND* and *FD* methods perform in a prominent way. Following the top plot in 5, *CHG* representation performs better than *HGR* in performing k-way clustering while takes slightly higher time to generate the required files.

Notably, in case of *GR* representation, k-way clustering times for all the methods are significantly lower comparing to both *HGR* and *CHG* representations although the graph file generation time is significantly higher than others. Similar demonstrations are also observed in the bottom plot of Figure 5 while the overall processing and computing time for all the activities have almost reduced to half. Moreover, *FDFND* and *FD* based methods perform in a much better way. This exhibits the effect of dynamically vary the targeted k value for desired number of clusters, thus signifies the efficacy of both *FDFND* and *FD* methods.

Figure 6 shows the aggregated computing times for all the concerning methods and representations with fixed and dynamic number of clusters. It clearly shows that the aggregated processing and computing time required for individual schemes decreases significantly with dynamic number of clusters comparing to using a fixed number of targeted clusters. In both the cases *GR* based representation perform well comparting to *CHG* and *HGR* ones. Again, *FDFND* and *FD* methods perform almost similarly for all the representations. However, considering the primary objectives of minimising the measure of dti, lb and dm, *FDFND* method performs well ahead of

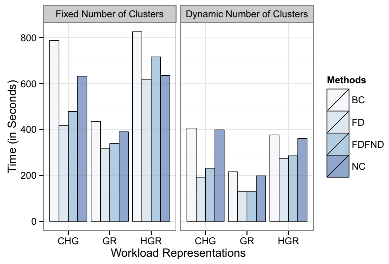


Fig. 6. Aggregated timing measure with fixed and dynamic number of clusters.

others, and produce the desired *best-effort* outcomes for the underlying OLTP database in incremental repartitioning.

V. Conclusion and Future Works

In this paper, a workload-aware incremental database repartitioning technique is proposed which effectively harness the power of progressive data analysis and stream mining to identify the most frequently accessed distributed transactional tuple sets with the aid of several proactive transaction classification techniques. From the experimental results it is clearly visible that the processing and computing time for large-scale workload networks can be reduced significantly by adopting intelligent data analysis methods in Big Data computing. Finally, from the experimental analysis with different workload network representations shows that dynamic adoption is required for such workload-aware approaches to effectively handle shared-nothing OLTP databases to guarantee acceptable level of DTs, load-balance, and physical data migrations. Our future works include using association rule mining from the observed transactional patterns to improve the classification performance in incremental steps.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “A view of cloud computing,” *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[2] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1986.

[3] G. Hohpe, “Your coffee shop doesn’t use two-phase commit,” *IEEE Software*, vol. 22, no. 2, pp. 64–66, Mar. 2005.

[4] L. Gu, D. Zeng, P. Li, and S. Guo, “Cost minimization for big data processing in geo-distributed data centers,” *Emerging Topics in Computing, IEEE Transactions on*, vol. 2, no. 3, pp. 314–323, Sept 2014.

[5] C. Curino, E. Jones, Y. Zhang, and S. Madden, “Schism: a workload-driven approach to database replication and partitioning,” *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 48–57, Sep. 2010.

[6] A. Quamar, K. A. Kumar, and A. Deshpande, “Sword: scalable workload-aware data placement for transactional workloads,” in *Proceedings of the 16th International Conference on Extending Database Technology*, ser. EDBT ’13. NY, USA: ACM, 2013, pp. 430–441.

[7] “Minimum k-cut,” http://en.wikipedia.org/wiki/Minimum_k-cut, [Online; accessed 31-October-2014].

[8] D. Kossmann, “The state of the art in distributed query processing,” *ACM Computing Survey*, vol. 32, no. 4, pp. 422–469, Dec. 2000.

[9] K. A. Kumar, A. Deshpande, and K. Samir, “Data placement and replica selection for improving co-location in distributed environments,” *Computing Research Repository (CoRR)*, vol. abs/1302.4168, 2013.

[10] K. Kumar, A. Quamar, A. Deshpande, and S. Khuller, “Sword: workload-aware data placement and replica selection for cloud data management systems,” *The VLDB Journal*, pp. 1–26, 2014.

[11] Z. Wei, G. Pierre, and C.-H. Chi, “Cloudtps: Scalable transactions for web applications in the cloud,” *Services Computing, IEEE Transactions on*, vol. 5, no. 4, pp. 525–539, Fourth 2012.

[12] A. Turk, R. O. Selvitopi, H. Ferhatosmanoglu, and C. Aykanat, “Temporal workload-aware replicated partitioning for social networks,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 11, pp. 2832–2845, Nov 2014.

[13] N. Xu, “Fine-grained data partitioning framework for distributed database systems,” in *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion*, ser. WWW Companion ’14. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2014, pp. 57–62.

[14] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu, “Mining frequent patterns in data streams at multiple time granularities,” in *Proceedings of the NSF Workshop on Next Generation Data Mining*, 2002.

[15] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, “Finding hierarchical heavy hitters in streaming data,” *ACM Transactions on Knowledge Discovery from Data*, vol. 1, no. 4, pp. 2:1–2:48, Feb. 2008.

[16] J. Cheng, Y. Ke, and W. Ng, “Maintaining frequent closed itemsets over a sliding window,” *Journal of Intelligent Information Systems*, vol. 31, no. 3, pp. 191–215, 2008.

[17] J. Guo, P. Zhang, J. Tan, and L. Guo, “Mining frequent patterns across multiple data streams,” in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM)*, New York, USA, 2011, pp. 2325–2328.

[18] D. Ienco, A. Bifet, I. liobait, and B. Pfahringer, “Clustering based active learning for evolving data streams,” in *Discovery Science*, ser. Lecture Notes in Computer Science, J. Frnkranz, E. Hllermeier, and T. Higuchi, Eds. Springer, 2013, vol. 8140, pp. 79–93.

[19] G. Cormode and M. Hadjieleftheriou, “Methods for finding frequent items in data streams,” *The VLDB Journal*, vol. 19, no. 1, pp. 3–20, Feb. 2010.

[20] “Distributed Transaction Processing: The XA Specification,” https://www2.opengroup.org/ogsys/catalog/c193, [Online; accessed 31-October-2014].

[21] “Roaming in GSM Network,” http://en.wikipedia.org/wiki/Roaming, [Online; accessed 31-October-2014].

[22] B. Mozafari, C. Curino, A. Jindal, and S. Madden, “Performance and resource modeling in highly-concurrent oltp workloads,” in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’13. New York, NY, USA: ACM, 2013, pp. 301–312.

[23] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, “Characterizing, modeling, and generating workload spikes for stateful services,” in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC ’10. New York, NY, USA: ACM, 2010, pp. 241–252.

[24] M. Quadrana, A. Bifet, and R. Gavaldà, “An efficient closed frequent itemset miner for the moa stream mining system,” *Artificial Intelligence Research and Development*, vol. 26, pp. 203–212, 2013.