2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing

Workload-Aware Incremental Repartitioning of Shared-Nothing Distributed Databases for Scalable Cloud Applications

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

Rajkumar Buyya
Dept. of Comp. and Information Systems
University of Melbourne
Victoria, Australia
Email: rbuyya@unimelb.edu.au

Abstract*—Cloud applications often rely on shared-nothing distributed databases that can sustain rapid growth in data volume. Distributed transactions (DTs) that involve data tuples from multiple geo-distributed servers can adversely impact the performance of such databases, especially when the transactions are short-lived in and require immediate response. The k-way min-cut graph clustering algorithm has been found effective to reduce the number of DTs with acceptable level of load balancing. Benefits of such a static partitioning scheme, however, is short-lived in Cloud applications with dynamically varying workload patterns where DT profile changes over time. This paper addresses this emerging challenge by introducing incremental repartitioning. In each repartitioning cycle, DT profile is learnt online and k-way min-cut clustering algorithm is applied on a special sub-graph representing all DTs as well as those non-DTs that have at least one tuple in a DT. The latter ensures that the min-cut algorithm minimally reintroduces new DTs from the non-DTs while maximally transforming existing DTs into non-DTs in the new partitioning. Potential load imbalance risk is mitigated by applying the graph clustering algorithm on the finer logical partitions instead of the servers and relying on random one-to-one cluster-to-partition mapping that naturally balances out loads. Inter-server data-migration due to repartitioning is kept in check with two special mappings favouring the current partition of majority tuples in a cluster—the many-to-one version minimising data migrations alone and the one-to-one version reducing data migration without affecting load balancing. A distributed data lookup process, inspired by the roaming protocol in mobile networks, is introduced to efficiently handle data migration without affecting scalability. The effectiveness of the proposed framework is evaluated on realistic TPC-C workloads comprehensively using graph, hypergraph, and compressed hypergraph representations used in the literature. Simulation results convincingly support incremental repartitioning against static partitioning.

Keywords*—*Cloud databases; workload; distributed transactions; incremental repartitioning; load-balance; data migration;

I. Introduction

Nowadays, electronic data are being generated in an unprecedented speed with the dynamic and planetary expansion in e-commerce, online business processing, digital media, and social networks. It is estimated that 2.3 trillion gigabytes of digitised data are generated everyday around the globe [1]. As an example, in an average day, over 30 billion pieces of contents are shared in Facebook while 4 billion hours of videos are watched in YouTube [1]. In recent years, such Internet-scale Web applications scale-out instantaneously using Cloud computing technologies. Shared-nothing distributed databases in combination with horizontal data partitioning, provide a key mechanism to handle this massive data explosion and to scale to billions of concurrent users. Unfortunately, traditional approaches can hardly adopt the dynamic workload characteristics without expansive data redistributions and load balance operations within a geo-distributed cluster [2]. With the dynamic nature of user-facing interactive Web applications driving Online Transaction Processing (OLTP) workloads, its simply not possible for a static partitioning and placement model to work effectively by only adding more servers and hard disks to the cluster. By nature, OLTP transactions are small-sized and short-lived with an immediate response time requirement. At the same time, within a partitioned database, DTs occur frequently and span across multiple servers creating unscalable communications in transaction processing [3]. In addition, to adopt dynamic workloads, large-scale data migrations are required involving significant cost in terms of I/O, database resources, and potential downtime.

Recently proposed techniques for workload-aware data partitioning [4], [5] monitor the transactional logs and periodically create workload networks using graph or hyper graph representation. Each edge in a workload graph connects a pair of tuples originated from the same transaction whereas a hyper edge connects all tuples within a transaction in a hypergraph. Later, these workload representations are clustered using k-way min-cut clustering, and then randomly placed across the set of physical servers within a database cluster. As long as workload characteristics do not change dramatically, and tuples from a cluster stay together in a physical server, the occurrences and adverse impacts of DTs are reduced rapidly. A number of centralised data lookup and routing mechanisms are also proposed to support such dynamic data redistribution. Large-scale OLTP service providers develop partition management solutions like YouTube's Vitess [6], Tumblr's JetPants [7], and Twitter's Gizzard [8] to deal with rapid data growth. Nonetheless, the underlying data placements are not transparent to application codes, and redistributions are not aware of workload dynamics. Furthermore, none of these techniques provide any explicit way to minimise physical data migrations over WAN, and global load balance at the same time for a geo-distributed shared-nothing database cluster.

In this paper, we present a proactive workload-aware incremental repartitioning framework which transparently redistributes database tuples to ensure minimum data migrations and global load balance. Transactional logs are collected periodically then undergo a pre-processing and classification stage before generating workload networks for min-cut clustering. A unique transaction classification process is introduced to identify purely distributed transactions and non-distributed ones containing *moveable* data tuples that are also contained in a DT. This novel classification removes the shortcomings of selective swapping of tuple sets for local load balancing by extending the size of workload network, and over the time reduces the impact of DTs ensuring global optimisation in both load balance and data migrations. We also perform a detail sensitivity analysis by representing the workload networks in fine, exact, and coarse granularity using graphs, hypergraphs, and compressed hypergraphs. In contrary to previous works, a fixed number of clusters are created from the workload network for the total number of logical data partitions in the entire database instead of the number of physical servers. This provides finer control in load balance over the set of both partitions and servers. We also avoid tuple-level replications to observe the quality of incremental repartitioning under worst-case scenario of DTs. We also propose two innovative cluster-to-partition mapping strategies that cater for minimising both physical data migrations and distribution imbalance. Our distributed data lookup mechanism ensures high-scalability, and guarantees a maximum of two lookups to locate a tuple within the partitioned database.

To evaluate the quality of incremental repartitioning, we devise a set of metrics and also provide a way to administratively direct a particular repartitioning objective using a composite metric. Finally, we compare the quality of the proposed incremental repartitioning framework against a static partitioning configuration similar to [4] implementing random one-to-one cluster-to-partition mapping with different workload representations. More specifically, we compare 12 different database configurations with different settings–3 workload representations and 3 mapping strategies for static and incremental repartitioning. Our simulation based experimental results using realistic TPC-C workload signify the trade-offs between different repartitioning approaches while showing the clear shortcomings of a static partitioning in achieving dynamic data redistributions for OLTP databases.

The main contributions are summarised in below:

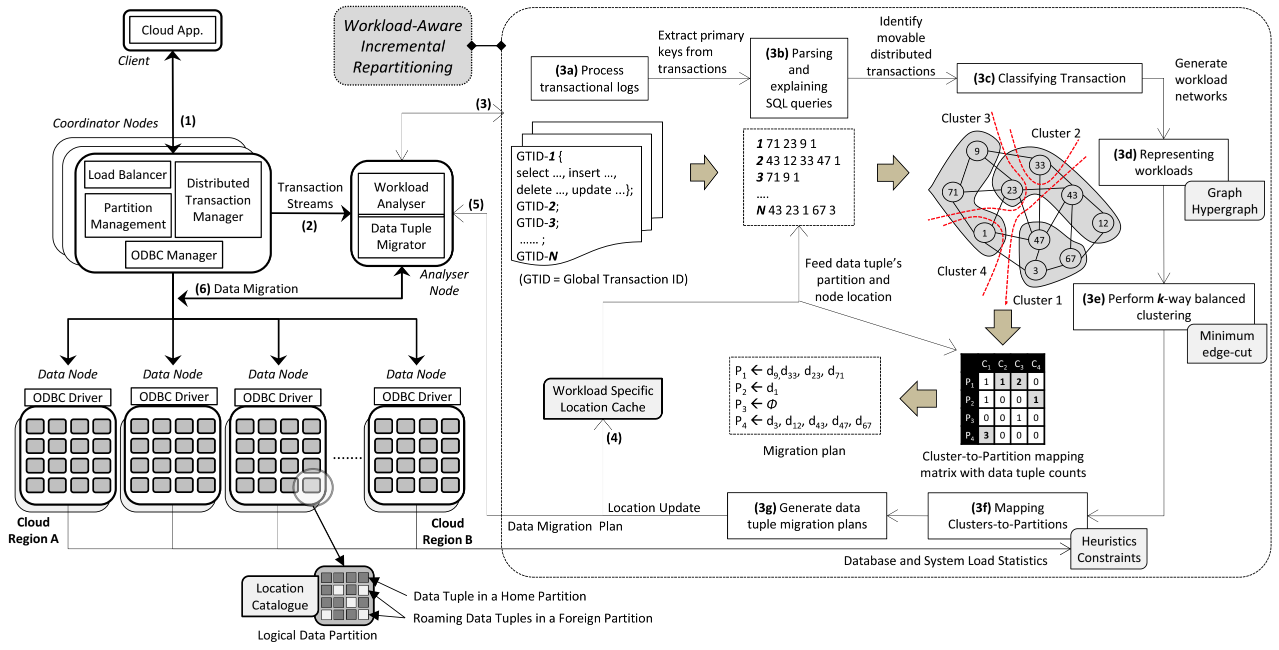
- Investigating possible design choices for workload network representations and their applicability.
- Proposing a proactive transaction classification technique that identifies DTs and moveable non-DTs to create workload networks.
- Presenting two cluster-to-partition mapping strategies that ensure minimum inter-server data migrations and load imbalance across partitions and servers.
- Developing a scalable distributed data lookup technique that requires a maximum of two I/O roundtrips to locate a data tuple within the entire database.
- Devising a set of quality metrics for the incremental repartitioning process defining different objectives.

The remainder of this paper is organised as follows: we review the related works in brief in Section II; a high-level overview of the proposed framework is discussed in Section III; Section IV details the steps, formulations, and design philosophies with necessary illustrations; Section V discusses the experimental results comparing to a static partitioning framework; and finally Section VI concludes the paper.

II. Related Work

Workload-aware load balance with I/O overhead minimisation in distributed database systems was studied before for finding optimal data placement strategy in shared-nothing parallel databases [9]. Recent works primarily focus on OLTP workloads for scaling-out the Cloud applications to minimise the number of DTs. Workload-aware data replication and partitioning approach is first introduced by [4] for OLTP databases. The authors proposed 'Schism' which represents the transactional workload as a graph, and performs k-way replicated graph partitioning to minimise the effect of DTs. However, 'Schism' usually generates very large graphs, does not deal with dynamic workload changes, and the more general problem of repartitioning. Transactional workloads are modeled as compressed hypergraph in [5] by hashing data tuple's primary key to reduce the overhead of k-way clustering. The authors propose 'SWORD', an incremental repartitioning technique which moves a fixed amount of data in a regular interval upon notifying workload changes, and by observing the increase in the percentage of DTs from a predefined threshold. However, this reactive approach only ensures local load balance, and does not always guarantees reduction in DTs. Due to the selective swapping of the randomly compressed tuple sets and newly transformed DTs, the quality of min-cut clustering may lost, and gradually lead to global data distribution imbalance. In [10], another automatic workload-aware database partitioning method is proposed along with an analytical model to estimate skew and coordination cost for DTs. It uses the same graph based workload representation of [4], and primarily focuses on optimal database design based on workload characteristics. However, it did not consider incremental repartitioning.

'Elasca' is proposed in [11], where a multi-object workload-aware online optimiser is developed for optimal partition placement ensuring minimum data movement, however it does not support incremental repartitioning. A distributed lookup method for transactional databases requiring special 'knowledge nodes' for coordination is proposed in [12], however it may perform incorrect routing due to inconsistent values. In contrast, our proposed distributed lookup operation is based on the well known concept of *roaming* [13], and it always guarantees consistent results with a maximum of two lookups. In [14], a Social Partitioning and Replication middleware–(SPAR) is proposed that explores the social network graph from user interaction, and then performs joint partitioning and replication to ensure local data semantics for the users. Similarly, in [15], temporal activity hypergraphs are used to model user interactions in social network, and then min-cut clustering is used to minimise the impact of DTs with minimum load imbalance. However, none of these techniques explore the incremental repartitioning problem, and the effect of data migrations in global load balance.



Cloud App.
Client
Workload-Aware Incremental Repartitioning
Coordinator Nodes
Load Balancer
Partition Management
Distributed Transaction Manager
ODBC Manager
Transaction Streams
Workload Analyser
Data Tuple Migrator
Analyser Node
(6) Data Migration
Data Node
ODBC Driver
Cloud Region A
Cloud Region B
(3a) Process transactional logs
Extract primary keys from transactions
(3b) Parsing and explaining SQL queries
Identify movable distributed transactions
(3c) Classifying Transaction
Generate workload networks
(3d) Representing workloads
Graph Hypergraph
(3e) Perform k-way balanced clustering
Minimum edge-cut
(GTID = Global Transaction ID)
Feed data tuple's partition and node location
Workload Specific Location Cache
Migration plan
Cluster-to-Partition mapping matrix with data tuple counts
(3f) Mapping Clusters-to-Partitions
Heuristics Constraints
(3g) Generate data tuple migration plans
Location Update
Data Migration Plan
Database and System Load Statistics
Location Catalogue
Logical Data Partition
Data Tuple in a Home Partition
Roaming Data Tuples in a Foreign Partition

Fig. 1. An overview of the workload-aware incremental repartitioning framework using numbered notations from 1-6 representing the overall work flow. Steps 3a to 3g represent the flow of workload analysis, representation, clustering, and repartitioning decision generation.

III. Proposed System Overview

An overview of the proposed framework is shown in Figure 1. We assume a set of *coordinator* nodes serving clients' requests, and manage the executions of transactional queries. Coordinators are connected with a set of geo-distributed *data* nodes where the logical partitions reside. Each logical partition contains a location catalogue where the residing tuples locations and their current partition ids are persisted as key-value pairs. Note that, individual data nodes can be synchronously replicated as master-slave within independent groups to ensure high availability which is a common deployment practice. Thus, in this work we do not explicitly handle tuple level replication like [4]. Coordinators also administer partition management operations (like split, merge, and migration) and incoming read/write workload balance. Streams of transactional logs are continuously pulled by the *analyser* node, and pre-process for analysis either in a time or workload sensitive window. Analyser node can also cache the most frequently appeared tuple location in a workload-specific catalogue which is kept updated upon inter-partition data migrations. Following Figure 1, the input of the workload-aware incremental repartitioning component (in dotted rectangle) is transactional logs, and the output is a partition-level data migration plan. The overall process has four primary steps:

Pre-processing, Parsing, and Classification. Client applications submit transactional queries in step 1, which is then processed by a *distributed transaction coordinator* that manages the distributed data nodes. Upon pulling the streams of transactional workloads in step 2, individual transactions are processed to extract the contained SQL statements at step 3a. For each SQL statement, the primary keys of individual tuples are extracted, and corresponding partition ids are retrieved from the embedded workload specific location catalogue in step 3b. In the classification process (3c), original DT and moveable non-DTs are identified along with their frequency counts in the current workload, and their associated costs of spanning multiple servers.

Workload Representation and k-way Clustering. In step 3d, workload networks are generated from the extracted transactional logs gathered in the previous step using graph or hypergraph. Tuple-level compression can further reduce the size of workload network. Since transactional graphs cannot fully represent transactions with more than two tuples using pair-wise relationship, we cannot directly minimise the impact of DTs in the workload. However, graph representations are much simpler to produce, and it adopted wide ranges of application specific usages that also help us to understand its importance in creating workload networks. On the other hand, hypergraphs can exploit exact transactional relationships, thus the number of hyper edge cuts exactly matches the number of DTs. Yet, popular hypergraph clustering libraries are computationally slower than the graph clustering libraries, and produce less effective results [4].

In reality, with the increase in size and complexity, both of these representations are computation intensive in manipulation. Furthermore, compression techniques can confine an algorithm within a specified target, dramatic degradation in clustering quality and overall load balance occur with a high compression ratio [5]. Finally, workload networks are clustered using k min-cut clustering employed by the graph and hypergraph clustering libraries in step 3e.

Cluster-to-Partition Mapping. At step 3f, a mapping matrix is created with the counts for tuples that are placed in the newly created cluster and originated from the same partition as the matrix element. The produced clusters from the min-cut clustering are then mapped to the existing set of logical partitions by following three distinct strategies. At first, we employ uniform random tuple distribution for mapping clusters to database partitions which naturally balances the distribution of tuples over the partitions. However, there is no proactive consideration in this random strategy for minimising data migrations. The second strategy employs a straight forward but optimal approach. It maps a cluster to a respective partition which originally contains maximum number of tuples from that cluster, hence minimum physical data migrations take place.

In many cases, this simple strategy turns out to be many-to-one cluster-to-partition mapping, and diverges uniform tuple distribution. Again, incremental repartitioning can create server hot-spot as similar transactions from new workload batches will always drive more new tuples to migrate into a hot server. As a consequence, overall load balance decreases over time, which is also observed in our experimental results. A way to recover from this situation is by ensuring that cluster-to-partition mapping remains one-to-one, which is used as the third strategy. This simple, yet effective, scheme restores the original uniform random tuple distribution with the constraint of minimising data migrations. Finally, in step 3g, based on different mapping strategies and applied heuristics a data migration plan is generated, and then forwarded to the data tuple migrator module in step 5.

Distributed Location Update and Routing. The analyser node keeps a workload specific location catalogue for the most frequently accessed tuples, and updates the associated locations at each repartitioning cycle in step 4. The analyser also directly invokes the corresponding data nodes to perform data migrations in step 6 without interrupting the ongoing transactional services. Until a tuple fully migrates to a new partition, its existing partition serves all the query requests. Distributed databases using range partitioning require keeping a central lookup table for the clients to retrieve tuples. Hash partitioning requires the client to use a fixed hash function to lookup the required tuples in the specified server. Consistent hash partitioning [16] employs distributed lookup mechanism using distributed hash table. However, none of these partitioning schemes provide scalable data lookup mechanisms for successive data redistribution.

To solve this problem, we use the well established concept of *roaming* from wireless telecommunications and computer data networks. The problem of location independent routing is already solved in IPv6 using Mobile IP [17], and in GSM networks using roaming mobile stations [13]. In a similar way, the attached location catalogue within each data partition keeps track of the *roaming* tuples and their corresponding *foreign* partitions. A maximum of two lookups are required to find a tuple without client-side caching. With proper caching enabled, this lookup cost can be even amortised to one for most of the cases with high cash hit.

IV. Workload-Aware Incremental Repartitioning

A. Problem Formulation

Let, $S = \{S_1, ..., S_n\}$ be the set of n shared-nothing physical database servers where each $S_i = \{\mathcal{P}_{i,1}, ..., \mathcal{P}_{i,m}\}$ denotes the set of m logical partitions reside in S_i. Again, let, $\mathcal{P}_{i,j} = \{d_{i,j,1}, ..., d_{i,j,|P_{i,j}|}\}$ denotes the set of tuples reside in $\mathcal{P}_{i,j}$. We can thus get the amount of tuples reside in S_i as $D_{S_i} = \bigcup_{\forall j} \mathcal{P}_{i,j}$. Finally, $D_S = \bigcup_{\forall i} DS_i$ denotes the total amount of tuples in the entire partitioned database.

TABLE I. Sample Database: Physical and Logical Layout

Servers	Partitions
S_1 (10)	P_1: (5)={2, 4, 6, 8, 10}
	P_3: (5)={12, 14, 16, 18, 20}
S_2 (10)	P_2: (5)={1, 3, 5, 7, 9}
	P_4: (5)={11, 13, 15, 17, 19}

Let, $\mathcal{W} = \{\mathcal{W}_1, ..., \mathcal{W}_\omega\}$ be the set of workload batches within the database lifetime τ. Each $\mathcal{W}_i$ represents a particular workload batch at the ith tick of τ where $\tau = \sum_{\forall i} \tau_i$. The set of transactions in any $\mathcal{W}_i$ is represented by $T = \{t_1, ..., t_z\}$, and can be either characterised as distributed (T_δ) or non-distributed (T_η), thus $T = T_\delta \bigcup T_\eta$ and $T_\delta \bigcap T_\eta = \phi$ where $T_\delta = \{t_{\delta,1}, ..., t_{\delta,|T_\delta|}\}$ and $T_\eta = \{t_{\eta,1}, ..., t_{\eta,|T_\eta|}\}$. Again, any distributed or non-distributed transaction $t_{\delta,i}$ or $t_{\eta,i}$ can occur multiple time within $\mathcal{W}_i$, hence, its frequency can be represented by either $freq(t_{\delta,i})$ or $freq(t_{\eta,i})$. As any $t_{\delta,i}$ can span multiple servers, we define the cost of spanning as $cost(t_{\delta,i})$. We consider the cost of spanning multiple partitions by a transaction within a server negligible in terms I/O overhead. Let's define the problem of incremental repartitioning as:

Problem Definition: For a given transactional workload $\mathcal{W}_i$ at ith observation, $\mathcal{S}$ homogeneous servers containing total $\mathcal{P}$ logical partitions, and a maximum allowed imbalance ratio ϵ, find an incremental repartitioning solution $\mathcal{X}_i$ from the output of a k-way balanced clustering ζ which minimises the mean impact of DTs in $\mathcal{W}_i$ and imbalance in $\mathcal{D}_S$ across partitions and servers having minimum inter-server data migrations.

In the following, we use illustrative examples using a simple database construction with 20 data tuples distributed using hash-partitioning over 4 logical partitions and 2 physical servers as shown in Table I. A sample workload batch with 7 transactions and corresponding data tuples are also shown in Table II. Finally, a detail illustration on how the cluster-to-partition mapping strategies work with different workload representations is shown in Figure 3.

B. Workload Modelling

We model the workload networks using three distinct representations. Firstly, graph representation (*GR*) produces fine-grain workload network although it is unable to fully capture the actual transactional relationship between different tuples. Yet, graph *min-cut* process can still generate high quality k-way clustering and minimises the impact of DTs, unless the overall graph size increases with workload variability, and adequate level of sampling is performed [4]. Secondly, hypergraph representation (*HGR*) generates most accurate, and exact workload networks thus also able to produce balanced clusters with min-cut hypergraph clustering. Moreover, from our empirical studies we found that, k-way min-cut balanced hypergraph clustering produces more consistent results in terms of achieving the repartitioning goals, and is also mentioned in [15]. Finally, compressed hypergraph representation (*CHG*) produces coarse-grain workload networks depending on the compress level. With lower level of compression, less coarse

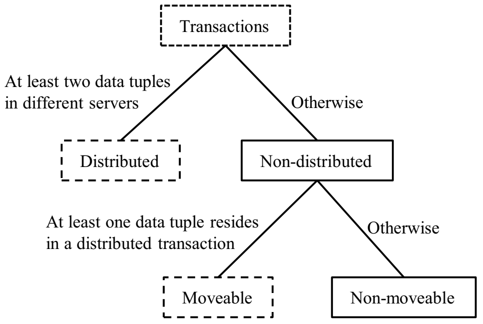

Transactions
At least two data tuples in different servers
Otherwise
Distributed
Non-distributed
At least one data tuple resides in a distributed transaction
Otherwise
Moveable
Non-moveable

Fig. 2. Transaction classification identifying DTs and moveable non-DTs.

networks are generated and k-way clustering performs better. However, as shown in [5], as the level of compression increases the quality of the clustering process degrades dramatically. We formally define the individual representations as in below:

1) Graph Representation: A graph $\mathcal{G} = (\mathcal{V}, E_g)$ represents $\mathcal{W}_i$ where each edge $e_g \in E_g$ links a pair of tuples (v_x, v_y) from $\mathcal{V} = \{v_1, ..., v_{|\mathcal{V}|}\} \subset \mathcal{D}_S$ for a transaction t_i where $v_i = \exists a \exists b \exists c\, d_{a,b,c}$. Individual tuples from (v_x, v_y) connects to their respective set of adjacent tuples $\mathcal{A}_{v_x}$ and $\mathcal{A}_{v_y}$ originated from the same t_i. Any edge within t_i has a weight representing the frequency of t_i in $\mathcal{W}_i$ which co-access the pair (v_x, v_y), while vertex weight represents the tuple's size (in volume).

2) Hypergraph representation: A hypergraph, $\mathcal{H} = (\mathcal{V}, E_h)$ represents $\mathcal{W}_i$ where a hyperedge $e_h \in E_h$ characterises a transaction t_i and overlays its contained set of tuples $\mathcal{V}_{t_i} \subset \mathcal{V}$. A hyperedge representing t_i is associated with a weight denoting the frequency of e_h within $\mathcal{W}_i$ and its vertices' weight represent data tuples' size (in volume).

3) Compressed Hypergraph representation: A hypergraph, $\mathcal{H} = (\mathcal{V}, E_h)$ can be compressed by collapsing the vertices to a set of virtual vertices $\mathcal{V}'$ using a simple hash function on the primary keys [5]. A compressed hypergraph $\mathcal{H}_c = (\mathcal{V}', E'_h)$ represents $\mathcal{W}_i$ where each virtual hyperedge $e'_h \in E'_h$ constitutes the set of virtual vertices $v'_{e_h} \subset \mathcal{V}'$ where the original vertices of e_h are mapped into and $|v'_{e_h}| \geq 2$. Virtual vertex weight represents the combined data volume sizes of the corresponding compressed tuples. And hyperedge weight represents the frequency of transactions which access the corresponding virtual vertices. C_l denotes the compression level as $|\mathcal{V}|/|\mathcal{V}'|$ and equals to 1 for no compression while to $|\mathcal{V}|$ for full compression.

Figure 3 presents the workload networks as graph, hypergraph, and compressed hypergraph (with $C_l = 0.5$) for the transactions listed in Table II.

C. Proactive Transaction Classification

In constructing the classification technique, we argue that there always exists a group of tuples which are retrieved while processing the DTs, and also participated in the execution of non-distributed but frequently occurred transactions. These particular groups of tuples when move into different database servers due to the database repartitioning process can turn the previously non-DTs into newly distributed ones. We use this intuitive to classify the workload transactions into three

TABLE II. SAMPLE WORKLOAD

Transaction	Data Tuples	Class
T_1	{1, 4, 5, 6, 7, 8, 10}	DT
T_2	{1, 4, 6, 9, 11}	DT
T_3	{9, 15, 17}	Moveable Non-DT
T_4	{9, 17}	Moveable Non-DT
T_5	{5, 7, 18}	DT
T_6	{15, 17}	Non-moveable Non DT
T_7	{2, 14, 16}	Non-moveable Non DT

different categories – distributed, non-distributed moveable and non-distributed non-moveable as shown in Figure 2. As an example, transactions T_1, T_2, and T_5 from the sample workload of Table II are identified as *distributed*, whereas T_3 and T_4 are labelled as *moveable non-distributed*. Finally, T_6 and T_7 are discarded as purely *non-distributed* transactions.

Clearly, a number of non-distributed moveable transactions will be remain protected within k-way clustering as the *min-cut* clustering always tries to preserve as much as transactional edges it could. As the tuples in these moveable transactions did not participate into any DTs, they are residing in isolation within the workload network. Thus, they are highly likely to be preserved together in the same cluster after k-way clustering. As an example shown in Figure 3, the non-distributed moveable transactions T_3 and T_4 containing tuples with id 9, 15, and 17 remain protected as non-distributed after performing k-way clustering with all of three workload representations using Metis [18] and hMetis [19] libraries.

If we added the DTs T_1, T_2, and T_5 in the workload subgraphs, then at the next incremental repartitioning phase T_3 and T_4 would have been appeared as DT. Since, tuple with id 9, which by this time would have been already moved to another partition located in a different physical server, would cause its associated transactions to become distributed. There exists a clear trade-off between the increase of size of the workload networks and achieved benefits. At one end, the smaller is the workload network, it will less computationally costly to process with respect to time and I/O. On the other hand, if we include all the workload tuples in the representations, it may reduce the impact of DT better than in a particular repartitioning cycle, but with the price of unwanted data migrations to create new DTs. By aggressively classifying the *non-distributed moveable* transactions, the quality of the overall repartitioning process increases as the impact of DTs decreases comparing to a static partitioning strategy as shown later in our experimental results.

D. k-way Balanced Clustering of Workload

Given $\mathcal{G}$ and a maximum allowed imbalance ratio ϵ, we can define the problem as find the k-way clustering $\zeta_{\mathcal{G}} = \{V_1, ..., V_k\}$ that minimises transactional *edge cut* with the *balance* constraint bounds by $(1 + \epsilon)$. Similarly, the k-way constrained and balanced clustering of $\mathcal{H}$ is $\zeta_{\mathcal{H}} = \{V_1, ..., V_k\}$ such that minimum number of hyper edges are cut having the imbalance ratio ϵ. Analogously, the k-way balanced clustering of $\mathcal{H}_c$ is $\zeta_{\mathcal{H}_c} = \{V'_1, ..., V'_k\}$ with an imbalance ratio ϵ aiming at minimum virtual hyperedge cuts. Note that, we denote k as the total number of logical partitions instead of the number of physical servers. From our empirical experiments we find

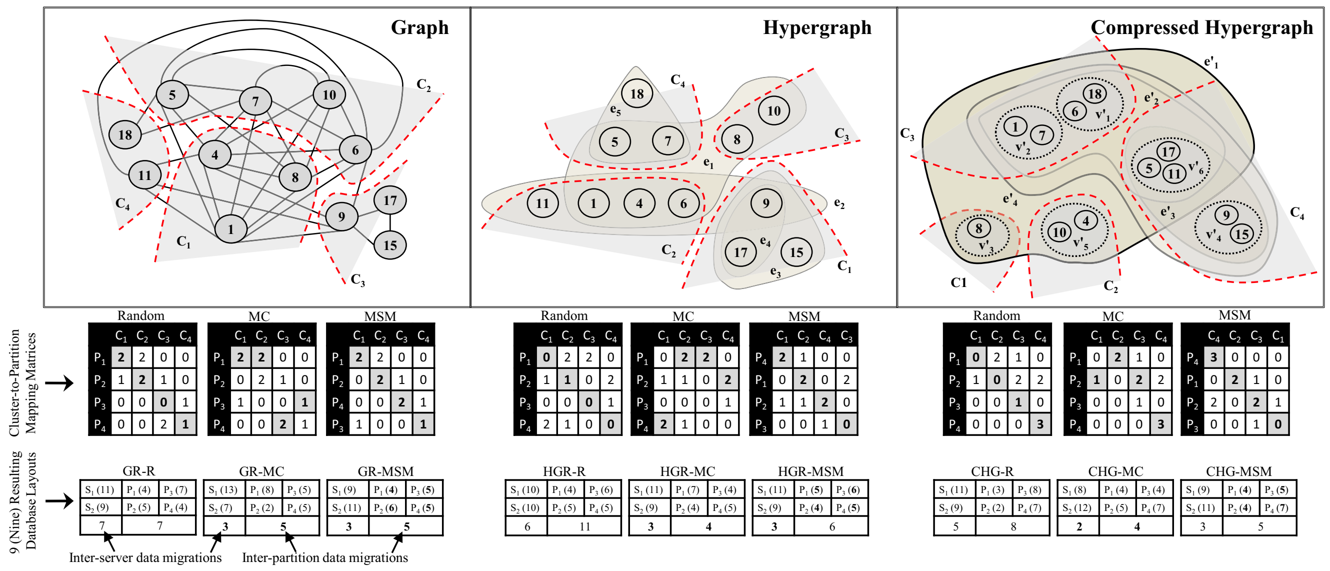

Graph
Hypergraph
Compressed Hypergraph
Random
MC
MSM
Cluster-to-Partition Mapping Matrices
9 (Nine) Resulting Database Layouts
GR-R
GR-MC
GR-MSM
HGR-R
HGR-MC
HGR-MSM
CHG-R
CHG-MC
CHG-MSM
Inter-server data migrations
Inter-partition data migrations

Fig. 3. Transactional workload modelling with 3 representations along with 4-way min-cut clustering, followed by 3 cluster-to-partition mapping strategies.

that executing the k-way clustering processing with k as the number of partitions provide finer granularity in balancing the distribution of data volume over the set of physical servers.

The k-way balanced clustering generates clusters of similar size with respect to the number of tuples given a *balance* constraint which is defined as $k \max(W(V_i))/W(V)$, and tells whether the clusters are equally-weighted or not. Here, $W(V_i)$ is the sum of the weights of the vertices in V_i. The partitions are said to be balanced if the *balance* measure is equals to or close to 1 otherwise imbalanced if greater than 1.

E. Cluster-to-Partition Mapping Strategies

Figure 3 presents three distinctive cluster-to-partition mapping strategies (in matrix format) beneath their respective workload network representations. The rows and columns of the matrices represent partition and cluster id respectively. Individual matrix element represents tuple counts from a particular partition which is placed by the clustering libraries under a specific cluster id. The shadowed locations in the mapping matrix with the counts in *bold face* represents the resulting decision block with respect to the particular cluster and partition id. Individual tables below the matrices represent the state of the physical and logical layouts of the sample database. The last row of these tables reveals the counts of inter- and intra-server data migrations for each of these nine representative database layouts. The *bold face* numbers in the layout tables at bottom denote most balanced distribution and least count for data migrations. In below we explain the main philosophies behind these mapping strategies in detail.

1) Random (R) Cluster-to-Partition Mapping: Naturally, the best way to achieve load balance in any granularity is to assign the clusters randomly. Clustering tools like Metis and hMetis randomly generates the cluster ids, and do not have any knowledge about how the data tuples are originally distributed within the servers or partitions. As a straightforward approach, the cluster ids can be simply mapped one-to-one to the corresponding partition id as they are generated. Although, this random assignment balances the workload tuples across the partitions it not necessarily guarantees minimum inter-server data migrations. As shown in Figure 3, the mapping matrices labelled with *Random* and database layouts with *GR-R*, *HGR-R* and *CHG-R* are the representatives of this class.

2) Max-Column (MC) Mapping: We aim at minimising the physical data migration within the repartitioning process using this strategy. In the cluster-to-partition mapping matrix the maximum tuple count of an individual column is discovered, and the entire cluster column is mapped to the represented partition id of that maximum count. Thus, multiple clusters can be assigned to a single partition. As maximum numbers of tuples are originated from this designated partition therefore they do not move from their home partition which reduces the overall inter-server physical data migrations. For OLTP workloads with skewed tuple distributions and dynamic data popularity, the impact of DTs can rapidly decrease from this greedy heuristic as tuples from multiple clusters may map to a single partition in the same physical server. However, this directly leads to data volume imbalance across the partitions and servers. Mapping matrices labelled as *MC* with corresponding database layouts of *GR-MC*, *HGR-MC*, and *CHG-MC* represent this mapping strategy in Figure 3.

3) Max-Sub-Matrix (MSM) Mapping: To both minimise load imbalance and data migrations, we fork lift the natural advantages of the previous strategies and combine them together. At first, the largest tuple counts within the entire mapping matrix are found and placed at the diagonally top left position by performing successive row-column rearrangements. The next phase begins by omitting the elements in the first row and column then recursively search the remaining *sub-matrix* for element with maximum tuple counts. Finally, all the diagonal positions of the matrix are filled up with elements having maximum tuple counts. Now, mapping the respective clusters one-to-one to the corresponding partitions results both minimum data migrations and distribution load balance. Note

TABLE III. COMPARISON OF SERVER AND PARTITION-LEVEL *balance*

Method	$S_{balance}$	$P_{balance}$
GR-R	1.1	1.4
GR-MC	1.3	1.6
GR-MSM	1.1	**1.2**
HGR-R	**1.0**	**1.2**
HGR-MC	1.1	1.4
HGR-MSM	1.1	**1.2**
CHG-R	1.1	1.6
CHG-MC	1.2	1.4
CHG-MSM	1.1	1.4

that, multiple maximum tuple counts can be found in different matrix positions, and the first such encountered element is chosen for simplicity.

The *MSM* strategy works similarly to the *MC* strategy as it prioritises the maximum tuple counts within the sub-matrices, and map the clusters one-to-one to the partitions like the *Random* mapping strategy thus preventing potential load imbalance across both the logical partitions and physical servers. In Figure 3, mapping matrices labelled as *MSM*, and representative database layouts *GR-MSM*, *HGR-MSM*, and *CHG-MSM* depict this mapping strategy.

F. The Balance Measure

For illustration purpose, we reuse the same *balance* measure [15] mentioned earlier while using the server and partition weights instead of cluster weights. Considering *GR-R* database layout as shown in Figure 3, there are total 20 tuples distributed among two physical servers (S_1, S_2) and four logical partitions ($P_1, ..., P_4$). These servers contain 11 and 9 tuples while the partitions contain 4, 5, 7, and 4 tuples respectively which leads to a *balance* value of $(2 \times 11)/(11 + 9) = 1.1$ at the server-level and $(4 \times 7)/(4 + 5 + 7 + 4) = 1.4$ at the partition-level. Table III presents the calculated *balance* measure for these entire nine database layouts in both server and partition-level where *bold face* values indicate lowest *balance* measure. In overall, *GR-MSM* and *HGR-MSM* perform better than all others primarily in terms of minimum data migrations and load balance. From this elaborate illustration, it is clear that k-way min-cut clustering of the workload network across partitions gives better estimation of load balance, and finer degrees of freedom for different cluster-to-partition strategies to minimise intra- and inter-server physical data migrations.

G. Distributed Data Lookup

As mentioned in Section I and III, any centralised lookup mechanism is always at risk to be the bottleneck in achieving high-availability and scalability requirements. We take a sophisticated approach to distribute the data tuple lookup process into individual database partition level. Thus, data migration operations are totally transparent to distributed transaction processing and coordination. By maintaining a key-value list of *roaming* and *foreign* data id with their corresponding partition id, individual partitions can answer the lookup queries. Tuples are assigned permanent *home* partition id for its lifetime when the database is initially partitioned using range, hash, or consistent hash [16]. *Home* partition id only changes while a partition splits or merges and these operations are overseen by the *coordinators* as shown in Figure 1, thus transparent to the lookup process. As the tuple locations are managed by their *home* partitions, data inconsistency are strictly prevented. Unless a tuple is fully migrated to another partition, and its *roaming* location is written in the catalogue, the old partition continue serving transactional processing.

When a tuple migrates to another partition within the process of incremental repartitioning, only its respective *home* partition needs to be aware of it. The target *roaming* partition will treat this migrated tuple as a *foreign* and updates its lookup table accordingly whereas the original *home* partition will mark this tuple as *roaming* in its lookup table and update its current location with the *roaming* partition's id. A lookup process always query the tuple's *home* partition to retrieve it. If the tuple is not initially found in its original location, the lookup table entry thus immediately informs the most recent location of the tuple and redirect the search towards the *roaming* partition. Thus, a maximum of two lookup operations can be required to find a tuple within the entire database.

Note that, the cost of physical data migration may increase while using such distributed lookup process. With a high probability individual data migrations in the incremental repartition process may involve running location update process up to three physical servers serving the *home* partition and two *roaming* partitions — current and target partitions. At present, we are investigating the implication of this cost, and how to include this in the formulation of quality measure.

H. Quality Measure for Incremental Repartitioning

In evaluating the performance of the incremental repartitioning, previous works [4], [5] only measure the percentage of reduction in DTs. However, this single measure fails to imply any meaning conclusion about how the impact of distributed transaction is minimised. Further, there are no measures for overall load balance and data migrations. We propose three independent metrics to measure the successive repartitioning quality achieving three distinct objectives – 1) minimise the impact of DTs; 2) minimise load imbalance; and 3) minimise the number of physical data migrations.

The first metric measures the impact within a scale of 0 to 1 associating the frequency of DTs and their related the cost of I/O. The second metric measures the tuple-level load distribution over the set of servers using *coefficient of variation* which effectively shows the dispersion of data load over successive period of observations. The third metric measures the mean inter-server data migrations for successive repartitioning processes. By combining all three aforementioned mentioned metrics, a composite metric is also proposed which represents the mix of workload representation and cluster-to-partition mapping strategy for a particular incremental repartitioning cycle to achieve a certain objective. In the following, we model these three representative metrics in detail.

1) The Impact of Distributed Transaction: Considering the formal definitions provided in Section IV-A, we combine the cost of spanning multiple physical server by any distributed transaction $t_{\delta,i}$, $cost(t_{\delta,i})$ with the frequency of $t_{\delta,i}$ within $\mathcal{W}_i$, $freq(t_{\delta,i})$. Here, $cost(t_{\delta,i}) = S_{t_{\delta,i}} = \{\forall v \in t : a \mid \exists a \exists b \exists c\, d_{a,b,c} = v\}$ which denotes the number of physical servers involved in processing $t_{\delta,i}$, whereas, $cost(t_{\eta,i}) = 1$ for any non-distributed transaction $t_{\eta,i}$. Note that, in reality this cost represents the overhead of I/O over the network while processing the DTs. Equation 1 below defines the spanning cost of T_δ within $\mathcal{W}_i$ for all $t_{\delta,i} \in T_\delta$

$$cost(T_\delta) = \sum_{\forall t_{\delta,i} \in T_\delta} cost(t_{\delta,i}) freq(t_{\delta,i}) \tag{1}$$

Similarly, (2) denotes $cost(T_\eta)$ for all $t_{\eta,i} \in T_\eta$

$$cost(T_\eta) = \sum_{\forall t_{\eta,i} \in T_\eta} freq(t_{\eta,i}) \tag{2}$$

Finally, the actual impact of T_δ can be defined as:

$$I_d = \frac{cost(T_\delta)}{cost(T_\delta) + cost(T_\eta)} \tag{3}$$

2) Load Balance: The measure of load balance across the physical servers is determined from the growth of the data volume with the set of physical servers. If we compute the standard deviation of data volume $\sigma_{\mathcal{D}_S}$ for all the physical servers, then, the variation of distribution of tuples within the servers can be observed. This is equivalent to what we discuss in Section IV-F as the *balance* measure. The coefficient of variation (C_v) defines the ratio between $\sigma_{\mathcal{D}_S}$ and $\mu_{\mathcal{D}_S}$ for all S under deployment, and independent of the unit of measurement. C_v can tell the variability of tuple distribution within the servers in relation to the mean data volume $\mu_{\mathcal{D}_S}$. Equation 4 below determines the C_v of the load balance measure for the entire cluster at any instance of observation.

$$L_b = \sigma_{\mathcal{D}_S} / \mu_{\mathcal{D}_S} \tag{4}$$

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) Inter-Server Data Migrations: For any given $\mathcal{W}_i$, the total of inter-server data migrations within τ_i can be normalised by dividing with the mean data volume $\mu_{\mathcal{D}_S}$. As shown in (5), D_m measures the quality of inter-server data migration with respect to the given workload $\mathcal{W}_i$.

$$D_m = M_v / \mu_{\mathcal{D}_S} \tag{5}$$

where M_v is the total number of migrations during the current observation window.

I. Composite Metric (C_m)

Let, C_m be the composite metric with weight factors ω_{I_d}, ω_{L_b}, and ω_{D_m} respectively for the objective measures I_d, L_b, and D_m where $\omega_{I_d} + \omega_{L_b} + \omega_{D_m} = 1$ providing two degrees of freedom to choose between different repartitioning goals. Besides, I_d, L_b, and D_m are further normalised by *0-1 normalisation* for unification purpose. Given the application and system requirements, system administrators can set specific goal towards achieving certain quality objectives – *minimise* I_d, L_b, or D_m for the incremental repartitioning process. Based on different weight distributions, it is thus possible to find a repartitioning sweet spot preferring particular choices of workload network representation and cluster-to-partition mapping strategy. Thus, by fine tuning the combinations in weight distribution one can instantly tackle unpredictable situations by tweaking the direction of incremental repartitioning process to maintain acceptable level of transactional services. We define the C_m according to the following equation:

$$C_m = \omega_{I_d} I_d + \omega_{L_b} L_b + \omega_{D_m} D_m \tag{6}$$

V. Experimental Results

To understand the effectiveness of the proposed ideas, we built a workload-driven simulation framework for the distributed database system presented in Figure 1. We evaluate our proposed methods against a static partitioning framework implementing the three workload network representations– graph, hypergraph, and compressed hypergraph using *random* cluster-to-partition mapping strategy. A workload-aware static configuration redistribute workload tuple only once and does not consider subsequent changes in transactional profile. This exhibits the worst-case scenario for an incremental repartitioning framework, and we consider it as the baseline of comparison. Tuple-level replication is not use in these settings as discussed in Section I and IV.

12 independent databases are compared using 3 workload network representations in combination of 3 cluster-to-partition mapping strategies. More specifically, we compare the databases *GR-R*, *GR-MC*, *GR-MSM*, *HGR-R*, *HGR-MC*, *HGR-MSM*, *CHG-R*, *CHG-MC*, and *CHG-MSM* as described through Section IV-B and IV-E against the static partitioning framework for all of the three individual *quality measures*. Our goal is to evaluate the effectiveness of the proposed techniques with respect to–I_d, L_b, and D_m (as detailed in Section IV-H) for incremental repartitioning cycles. Hence, we do not compare the results against the *performance measures* like transactional throughput and latency.

A. Experimental Setup

We use a realistic experimental setup of a distributed database as depicted in Figure 1, and use the popular TPC-C transactional workloads developed in our workload-driven simulator. A typical TPC-C database contains 9 tables, 5 transactions and simulates an order processing transactional system within geo-distributed districts and associated warehouses. Among these 'Stock' and 'Order-Line' tables are exceptionally fat in volume, and thus all the logical database partitions are not homogeneous in size. New tuples are inserted into 'Order' and 'Order-Line' tables using the 'New-Order Transaction' which usually occupies nearly 44.5% of the workload.

Fixed number of transactions are generated under 5 transaction types in each workload batch having a fixed birth and death rate. We further hash partition these 9 database tables by their primary ids, and place them into 10 data node servers having a total of 90 logical partitions. Note that, it is possible to hash partition TPC-C tables by 'Warehouse' id to balance the workload distribution, however we intentionally avoid this to exhibit the worst case scenario of DTs in a popular OLTP benchmark. The five types of transactions are weighted from heavy to light in terms of transactional processing, and they occur in high to low frequencies. The synthetic data generation process follows Zipf's distribution for generating 'Warehouse' and 'Item' tables, and use the database relationship constraints to generate others. We use Metis [18] and hMetis [19] k-way min-cut clustering libraries with their default settings. The entire simulation process runs for 10 times having 100

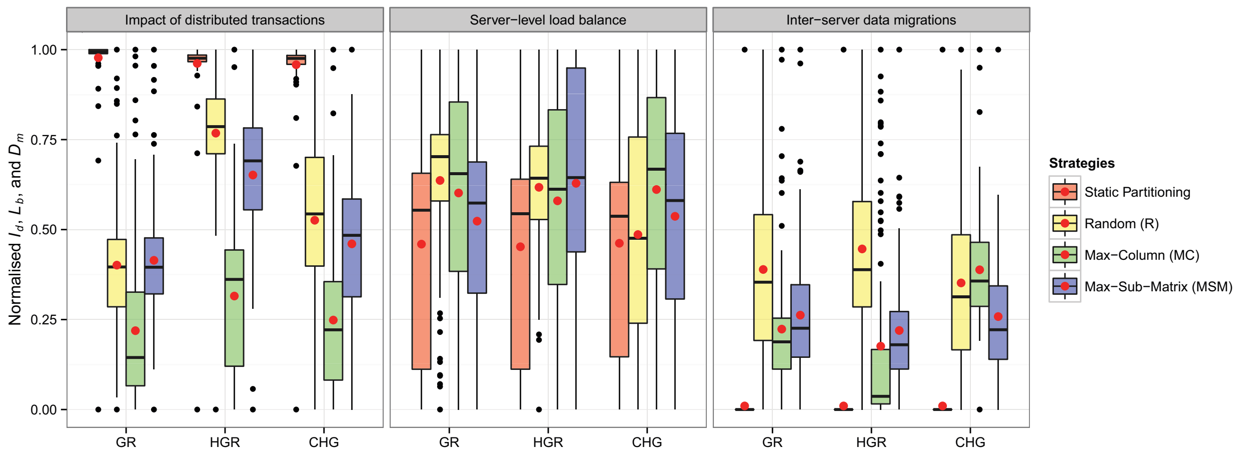


Fig. 4. Individual effects of I_d, L_b, D_m in incremental repartitioning comparing with the static partitioning scheme under different workload representations.

incremental repartitioning cycles in each run for all the 12 representative database configurations, and then averaged.

B. Result Analysis

1) Independent Measure of I_d, L_b, and D_m: Figure 4 presents the comparative results of the independent repartitioning quality measures for the 12 database configurations under test. Results are shown using box plots with small 'red circles' denoting *mean* values. Individual *quality measures* for different settings are grouped together into separate boxes. Within each box, 4 different strategies–1) *Static Partitioning*, followed by 3 Incremental Repartitioning approaches using 2) *Random (R)*, 3) *Max-Column (MC)*, and 4) *Max-Sub-Matrix (MSM)* cluster-to-partition mapping strategies (explained in Section IV-E) are compared. We expect the *Static Partitioning* scheme to exhibit the worst-case scenarios for all the individual metrics as it perform the workload-aware data redistribution only once, and do not run for the rest of the remaining 99 cycles. As shown in Figure 4, the values of I_d are too high for all the workload representations, load balance varies within a wide range of L_b, and mean data migrations are almost zero.

In evaluating I_d using (3), databases with *MC* based mapping strategies outperform all others due to aggressive data migrations and many-to-one cluster-to-partition mapping. For the very reason, data volume distributions of *GR-MC*, *HGR-MC*, and *CHG-MC* lead to complete imbalance over the 100 repartitioning cycles in each run. *MSM* databases performs somehow similar comparing to the *Random* strategy implementations. Although *graph* and *compressed hypergraph* based *GR-MSM* and *CHG-R* perform better, but in overall both *HGR-R* and *HGR-MSM* show good results without leaving few or any outliers at all. While comparing the results of L_b (4), both *Random* and *MSM* based databases perform well. Although, *HGR-MSM* shows much stable results in comparison to others, *GR-MSM* wins over all of them. This supports our intuition that randomness and one-to-one cluster-to-partition mapping can naturally balance data distributions across the database cluster. Inter-server data migrations (D_m using (5)) are specifically low with the *MC* databases, however, *GR-MSM* and *HGR-MSM* both performs reliably in terms of I_d, L_b, and D_m over successive repartitioning cycles showing the effectiveness of our proposed techniques. In *CHG* based configuration, *CHG-MC* performs better than *HGR-MC* for all the quality measures, however, *HGR-MSM* outperforms *CHG-MSM* in all aspects. While ranking the measures of these three individual metrics accordingly, *GR-MSM* wins over all other following by *CHG-MC* and *GR-MC*, while *HGR-MSM* achieves fourth place. Both of our proposed cluster-to-partition mapping strategies performs significantly better than the *Random* configuration in terms of minimising inter-server data migrations and load imbalance. In overall, the baseline–*Static Partitioning* scheme is found ineffective in handling dynamic OLTP workloads, and justifies our comprehensive studies with *Incremental Repartitioning* having different combinations of workload representations and mapping strategies.

2) Combined Effect Using Composite Metric, C_m: To understand the combined effect of I_d, L_b, and D_m through the composite metric C_m using (6), we use different combinations of the respected weight factors providing that $\omega_{I_d} + \omega_{L_b} + \omega_{D_m} = 1$. Figure 5 shows the resulting measure of C_m in a 2-d perspective plot using coloured scale where L_b and I_d are plotted in the X-axis and Y-axis respectively. The locations presenting the values of D_m can be determined by calculating $1-(\omega_{I_d}+\omega_{L_b})$ in the individual subplots. We can set specific preferences to prioritise one particular repartition *quality measure* over other. Individual extremes of I_d, L_b, and D_m can be found at (1, 0), (0, 1), and (0, 0) locations. By following the colour codes from the legend, one can easily identify how individual repartitioning objectives would be met. From the plots, as anticipated in Section IV, *MC* based databases do not favour L_b while dramatically reducing I_d in contrary. We can also identify the repartitioning choices for general-purpose OLTP application as *GR-MSM* and *GR-R* followed by *CHG-MSM* and *CHG-R*, while all of the *HGR* based settings are highly tunable depending on the repartitioning objectives in response to different administrative situations. A key observation here is that, the choices of workload representation and mapping strategy are not bounded to any specific combination. To confirm this, we also conduct two-way ANOVA test and

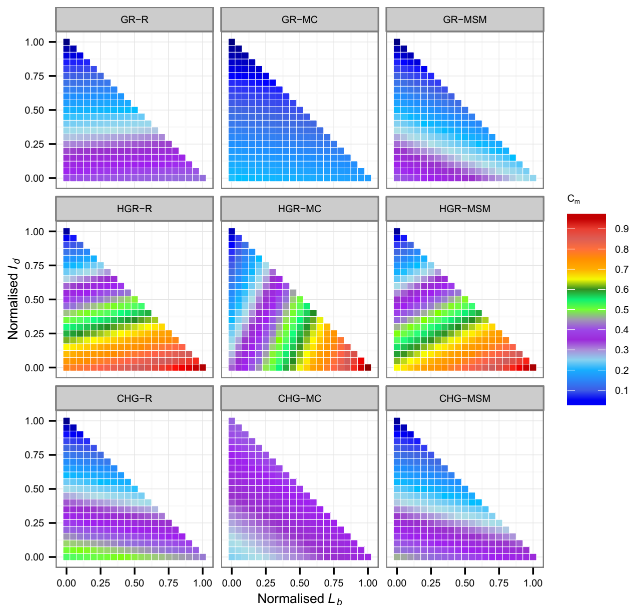


Fig. 5. Combined effect of I_d, L_b, and D_m through composite metric C_m. Note that, lower values of C_m indicate better solutions.

analyse the interaction plots. However, we did not find any true evidence of interactions between the choices of representation and mapping strategy. Results from ANOVA table also support this finding. These series of observations strongly support our arguments presented in Section III and IV, and justifies the goal of sensitivity analysis within a broad design space, which, to best of our knowledge was not done before.

VI. Conclusion and Future Works

In this paper, we present a workload-aware incremental repartitioning framework for OLTP databases which minimises – 1) the impact of DTs using the k-way balanced min-cut clustering; 2) the overall load imbalance through the randomness of the one-to-one cluster-to-partition mapping strategies; and 3) the physical data migrations by applying heuristics. Our innovative *transaction classification* technique ensures global minimisation in overall load imbalance and data migrations comparing to the worst-case scenario of a Static Partitioning framework implementing random cluster-to-partition mapping for different workload representations. The *elaborate modelling* approach clearly identifies the inter-related goals within the repartitioning process, and provides effective heuristics to achieve them based on operational requirements. By adopting the concept of *roaming*, the proposed distributed data lookup technique transparently decentralise lookup operations from the distributed transaction coordinator guaranteeing high-scalability. Our philosophical arguments broaden the decision space with comprehensive *sensitivity analysis* by combining different workload representations and mapping strategies. The proposed set of *quality metrics* presents a sophisticated way to measure the quality of successive repartitioning, and our simulation results outperform the Static Partitioning strategies in achieving individual repartitioning objectives. The use of *composite metric* shows an effective way of operational intelligence for Cloud applications suffering from dynamic workload behaviours. At present, we are investigating the followings as our future direction – incremental repartitioning enabling replication in a single step, enforce different *balance* criteria in the cluster-to-partition mapping heuristics, and include the cost of data migrations in modelling the *quality* measures.

References

[1] “IBM Data Hub,” www.ibmbigdatahub.com/infographic/four-vs-big-data, [Online].

[2] L. Gu, D. Zeng, P. Li, and S. Guo, “Cost minimization for big data processing in geo-distributed data centers,” *IEEE Transactions on Emerging Topics in Computing*, vol. 99, no. PrePrints, 2014.

[3] R. Johnson, I. Pandis, and A. Ailamaki, “Eliminating unscalable communication in transaction processing,” *The VLDB Journal*, vol. 23, no. 1, pp. 1–23, Feb. 2014.

[4] 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.

[5] 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.

[6] “Vitess – scaling MySQL databases for large scale web services,” https://github.com/youtube/vitess, [Online].

[7] “MySQL toolkit for managing billions of rows and hundreds of database machines,” https://github.com/tumblr/jetpants, [Online].

[8] “A flexible sharding framework for creating eventually-consistent distributed datastores,” https://github.com/twitter/gizzard/, [Online].

[9] M. Mehta and D. J. DeWitt, “Data placement in shared-nothing parallel database systems,” *The VLDB Journal*, vol. 6, no. 1, pp. 53–72, Feb. 1997.

[10] A. Pavlo, C. Curino, and S. Zdonik, “Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems,” in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’12. NY, USA: ACM, 2012, pp. 61–72.

[11] T. Rafiq, “Elasca: Workload-aware elastic scalability for partition based database systems,” Master’s thesis, University of Waterloo, Canada, May 2013. [Online]. Available: http://uwspace.uwaterloo.ca/handle/10012/7525

[12] B. P. Swift, “Data placement in a scalable transactional data store,” Master’s thesis, Vrije Universiteit, Amsterdam, Netherland, Feb. 2012. [Online]. Available: http://www.globule.org/publi/DPSTDS_master2012.pdf

[13] “Roaming in GSM Network,” http://en.wikipedia.org/wiki/Roaming, [Online].

[14] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez, “The little engine(s) that could: Scaling online social networks,” *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 4, pp. 1162–1175, Aug. 2012.

[15] 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. 99, no. PrePrints, 2014.

[16] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, “Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web,” in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, ser. STOC ’97. NY, USA: ACM, 1997, pp. 654–663.

[17] “Mobile IP,” http://en.wikipedia.org/wiki/Mobile_IP, [Online].

[18] G. Karypis and V. Kumar, “Multilevel k-way partitioning scheme for irregular graphs,” *Journal of Parallel and Distributed Computing*, vol. 48, no. 1, pp. 96–129, Jan. 1998.

[19] ——, “Multilevel k-way hypergraph partitioning,” in *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, ser. DAC ’99. NY, USA: ACM, 1999, pp. 343–348.