Can Graphs Solve the Geo-aware State Deployment Problem?

Diogo Lima[1,3], Hugo Miranda[1], François Taïani[2]

[1] LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal
dlima@lasige.di.fc.ul.pt

[2]Université de Rennes 1, IRISA, ESIR, Rennes, France
hamiranda@ciencias.ulisboa.pt

[3]Escola Superior de Hotelaria e Turismo do Estoril, Portugal
francois.taiani@irisa.fr

Abstract. A challenge for large scale distributed mobile applications, such as augmented reality games and social networks, is the management of application state. Because the infrastructure is concentrated in datacenters geographically distant from the end users, state flowing between the large number of users and the supporting infrastructure suffers from latency and network congestion degrading user quality of experience.
To mitigate this problem, *Fog Computing* proposes a physical approximation of the servers to the end users, avoiding the use of the Internet backbone. However, gains depend of the system capability to correctly deploy each state component at its most convenient location. Geo-aware state deployment is challenging as it requires a continuous observation and interpretation of the state utilization patterns, expected to evolve with time as new trends and user behavior emerge. This paper proposes a new object graph-based approach for the geo-aware state deployment problem and compares its performance with approaches based on linear algorithms.

1 Introduction

Large scale distributed mobile applications such as smart cities, augmented reality games (e.g. Ingress, Pokemon Go) and social networks (e.g. Foursquare) concurrently connect a large number of participants. Although with distinct degrees of involvement, all manipulate and exchange significant amounts of application state, mostly mediated by the application's supporting infrastructure. The current trend tends to concentrate this infrastructure in Cloud datacenters, implementing a geographical barrier between the end users and the managers of the application state. The latency and jitter that unavoidably results from this distance negatively impacts the application performance.

The Fog Computing approach [1] proposes an approximation of the servers to the end users by deploying surrogates at the network edge, in particular, on access networks. Although this approach has the potential to mitigate jitter and latency and improve performance, it depends on the ability to correctly deploy each software and state component at its most convenient location.

In the particular case of partial application state, a good geo-aware deployment depends of a successful identification of the state utilisation patterns. This is challenging, knowing that each part of the state is expected to exhibit a distinct pattern. As an example, consider a smart-city traveller support service. The predicted time of arrival of each bus is information more likely to be accessed by commuters waiting at the particular bus stop. However, the number of tickets available in the commuter's monthly pass is state expected to follow the commuter. Finally, the number of passengers waiting to travel on one specific bus route is state best managed at some location close to the bus company facilities. The problem is amplified by the consistency requirements that can be found between elements whose ideal location is distinct. A good example would be the average speed and number of passengers on all the buses in a single route.

An efficient implementation of geo-aware state deployment must implement policies that adapt to the observed utilisation patterns considering the most typical location but also the cost of guaranteeing consistency. In addition, the policies must consider the latency induced by the relocation of any component, which will necessarily disturb ongoing distributed transactions. Our problem is in contrast with existing caching solutions, whose main focus is to accelerate the access to replicas of shared data with a read-only status.

This paper approaches the problem of geo-aware state deployment using graphs partitioning algorithms. Originally proposed to partition database items across storage nodes [2], graph partitions are an interesting approach to distribute state components across different locations so that most of the transactions only need to access one. Unfortunately, the original approach does not consider geographical distribution and the impact of state components relocation. To address this limitation, the paper proposes a new object graph-based approach and compares its performance with others inspired on linear algorithms. The goal is to understand if graph partitioning can be used to identify correlations and to evaluate their suitability for geo-aware state deployment frameworks.

2 Problem Statement

The problem addressed in this paper assumes a large scale distributed application where components of the state can be correlated with geographical locations or with mobile users. In line with the Fog Computing model, we assume that our distributed application is supported by hardware located on a datacenter and a number of surrogates: servers located at the edge of the network. Surrogates provide storage and computing power, are connected to each other and to the cloud datacenter through a high speed wired network.

End users devices include desktops, mobile devices and sensors using high speed wired or wireless networks. It is expected that surrogates perform most of the computational effort, in particular, by application clients in their vicinity. Expectations are that most of the traffic produced by clients is handled at surrogates and can be replied without forwarding requests to the datacenter.

To address these expectations, the system must implement a geo-aware state deployment policy, allowing surrogates to store state at its location of interest.

In this model, the geo-aware deployment is managed by a service that decides in run-time the most suitable location of each state component and coordinates its transfer from one location to another. Conceptually, this service acts as an oracle, becoming eventually aware of all state updates and the location of the requester. Awareness is implemented in practice by having surrogates and datacenter servers to forward in the background the logs of operations they perform.

To tolerate faults, surrogates form partial replication groups that replicate their content. To reduce latency and facilitate consistency operations, partial replication groups membership follows a proximity criteria. For simplicity, our model equally considers an "umbrella" global replication group that includes all the surrogates and the datacenter. This global replication group is expected to keep state components widely accessed.

We assume that application state is composed of data that can be either unique to each user or device (personal data), collaborative data relevant to a specific geographical location (geo-aware data), or general application logic data (global data). Depending on the number of partial replication groups involved, state updates can be either *local* or *global*. Local operations are those that access data enclosed on a single partial replication group, while global operations are those where the data accessed is stored in multiple partial replication groups. It is assumed that the surrogate hosting the client requesting the state update participates in the operation.

The complexity involved in the coordination of global operations motivates the oracle to keep commonly accessed state components in the same partial replication group. Each migration decision must consider the cost of the migration itself, the benefits of increasing the number of local transactions and the cost of the global transactions to occur in the future.

3 Related Work

Devising efficient geo-aware deployment algorithms can leverage on previous research results on database partitioning and geo-replicated cloud storage. It should be noted that the problem goes beyond what is tipically addressed by Content Delivery Networks (CDN) like Akamai [1], given that clients of our target applications are expected to manipulate state.

Data partitioning was originally proposed as a response to the scalability and performance needs of database management systems. Its objective is to make a database more manageable by partitioning data into different nodes. Horizontal and vertical partitioning were the first policies to emerge. In horizontal partitioning, the rows of each table are evenly distributed by the partitions. Vertical partitioning follows the same approach but distributing columns by partitions. The rationale behind these approaches is to make the amount of data more manageable while making respectively the full set of attributes of the same rows or

[1] https://www.akamai.com/

the same attributes of all rows in each group. Other mechanisms, sharing partition data across shared-nothing servers have been proposed for example in [3–5]. These share our goal of minimizing the number of distributed transactions. However, they assume that all partitions are geographically collocated, thus ignoring the negative impact of distance on latency.

In contrast with the geo-oblivious approaches pursued by the databases community, cloud storages bring the notion of physical geography to data partitioning and proposed solutions at different scales. At a server-level scale, geo-replicated cloud storages aim to reduce latency and improve load balancing. At a datacenter scale, the goal is to provide fault-tolerance for datacenter level outages. AdaptCache [6] is a server-scale approach that proposes a cooperative and integrated cache framework for web enterprise systems where application servers cooperatively share their caches. Like in our approach, AdaptCache's "oracle" dynamically evaluates and manages object placement. However, it goes a step further and distributes requests across servers to achieve load balancing and facilitate consistency management. This approach is facilitated by the collocation of servers which, in contrast with our system model, permits AdaptCache to ignore the location of the clients. SPANStore [7] is a geo-replicated key-value store that unifies storage available in multiple datacenters into a single framework. The goal is to take advantage of pricing discrepancies to reduce the overall operation cost. This is in contrast with our model which assumes a dynamic setting where the "cost" of each object is represented directly by latency and indirectly by the distance between the client and the server storing the object.

Resilience to datacenter-level outages is addressed for example using distributed transactional SQL databases like CockroachDB [2]. In CockroachDB replica location is decided automatically, based on user configured criteria such as the types of failures to tolerate and the distinct locations made available by the user. Unfortunately, CockroachDB cannot encompass any concerns related with client access latency neither the dynamism associated to mobile applications.

Regardless of the data partitioning policy used, transactions that involve multiple partitions are by themselves a complex subject that deserves some attention. These tend to negatively impact system performance and several solutions try to mitigate their effect in the system. The approach in DS-SMR [8] evaluates the usage patterns to shape the data mapping. Overall, the goal is to design a dynamic and scalable state machine replication to exploit workload locality. Like in our approach, data partitioning is managed by an oracle. However, state reconfiguration follows an oversimplified rule that trades-off latency by the avoidance of distributed transactions and which consists in transferring the data to a single partition prior to initiating the global transaction. Workload analysis and location manipulation can also be found in the geo-replicated storage system SDUR presented in [9]. This paper shows that performance can be improved by giving priority to transactions using a single partition.

[2] https://www.cockroachlabs.com/

4 Geo-aware State Deployment Strategies

Anticipating the future utilization of state items is challenging and can only be efficiently determined if their utilization follows some pattern. Recall from Sec. 2 that in our system model the mapping of state items in surrogates is managed by a dynamic service implemented as an oracle. The oracle collects transaction logs from the surrogates to evaluate the most suitable one to host each state item. This paper reports on the results obtained with graph partitioning algorithms to be executed at the oracle. The goal is to understand the application and environment characteristics where these algorithms outperform other approaches, hereafter named linear algorithms.

Overall, the performance of the algorithms is evaluated considering two metrics. A perfect algorithm would minimize the number of state item transfers between surrogates and maximize the number of transactions that can be performed locally, i.e. using a single surrogate. The following sections present and discuss the algorithms according to their expected performance.

4.1 Linear Algorithms

Linear algorithm evaluate state items independently, i.e. decisions of where each state item must be placed are taken disregarding their participation on transactions with other state items. This paper evaluates the following linear algorithms:

Static (Static). In the Static algorithm, each object is assigned to the surrogate where it was first used and never changes. Considering the performance metrics above, Static trivially minimizes the number of surrogate state items migration at the cost of penalizing latency during distributed transactions. Therefore, it can be considered suitable for scenarios where the cost of distributed transactions is minimal in comparison with the cost of migrating state items.

Move to Next (MtN). At the beginning of each transaction, all state items involved are transferred to the surrogate that initiated the transaction. Items remain hosted on that surrogate until the algorithm is reapplied. MtN is a direct application of DS-SMR [8] and minimizes the number of distributed transactions at the cost of an expectedly large number of item migrations among surrogates.

Aggressive Polling (AP). Items are moved to the surrogate where they have been more used. It is assumed that the algorithm has an infinite memory, what can represent a penalty for the quick reaction to usage pattern changes for long standing state items. However, it is expected to present a suitable balance between the two radical approaches above avoiding spurious state item transfers.

4.2 Graph-partitioning Algorithms

Graph partitioning algorithms aim to improve performance by considering the association between state items in transactions. This is expected to favour the observation and adequate reaction to correlations and clusters, a capability that has been previously reported in [2]. The paper presents a graph-based approach to partition database items across storage nodes so that most transactions only need to access one node. In this approach data items are represented by vertices, connected with an edge if they are accessed together by at least one transaction. Vertex weight is used to represent the absolute access frequency of an object, while edge weight represents the number of transactions that accessed both objects. The graph is then partitioned into k partitions by graph partitioning algorithm such as k-way min/cut [10]. This algorithm focus on minimizing the edge-cut (the objective) in a way such that the sum of the vertex-weight in each partition is the same (the constraint). The minimization of the edge-cut cost favors the deployment of vertices frequently accessed in conjunction in the same partition. On the other hand, the vertex-weight constraint is necessary to avoid the trivial solution of keeping all vertices in a single partition and paying a edge-cut cost of 0.

*Hotspot Weighted Graph (*HWG*).* The graph representation described is unable to represent geographical distribution, crucial for our system model. This is an important aspect as more populated areas may access more state items than others. In our scenario, partition imbalances are expectable and should be preserved if they represent the effective utilization pattern. To address this problem, the HWG algorithm adds a new "surrogate" vertex type which reflects the association of a state item to the surrogate. For the edge-cut algorithm, "surrogate" vertexs are indistinct from the remaining, with edge weight reflecting the number of transactions using the state item initiated in the surrogate. Surrogate vertices are given a weight large enough to force the partition algorithm to deploy one at each partition. All other vertices were assigned a weight of 0, to encourage the algorithm to focus on edge-cut cost and disregard the leveraging of the number of state items per partition (i.e. load-balancing).

To better illustrate this model, Fig. 1 depicts Twiter hashtags distribution by 4 surrogates (Lisbon, Porto, Braga and Faro) after HWG is executed (details are presented in Sec. 5). The figure evidences 4 large hotspots, each representing one surrogate vertex and the association of each vertex (the tweet hashtag) to the surrogates where the tweet was performed. Tweets at the center of the graph are connected to more than one location.

4.3 Deployment Revision Frequency

The frequency of execution of the deployment algorithm can have a non negligible impact on the system. Each run of the algorithm will necessarily require computing power at the oracle and have the negative impact on performance of migrating state items between surrogates. However, algorithm executions are crucial to ensure the proper deployment of the state items and reduce the number of transactions envolving two or more surrogates. In this paper, a moderate approach of running the algorithms after every 1000 transactions was adopted.

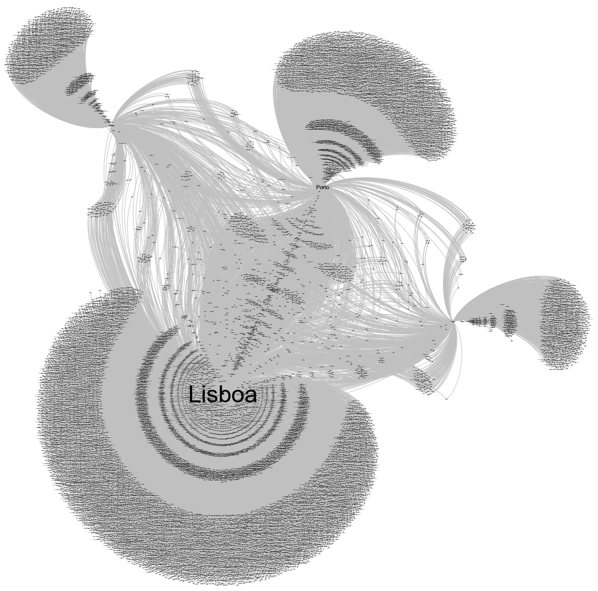

Fig. 1: Geographical object graph representation of a Twitter dataset from 2017 in Portugal.

It should be noted that the decision to use a frequency of 1000 transactions considers results observed in preliminary experiments with very frequent revisions. These experiments showed that as the revision frequency increases, so does the probability of the algorithms to follow inconsistent utilisation patterns, observed in the last few transactions. As a result, the number of state items migration increases exponentially, negatively impacting system performance.

5 Evaluation

To understand the impact of our state deployment strategies on system performance, we prepared an evaluation scenario using Twitter. The 53392 tweets used in this study were collected using Twitter's Streaming API,[3] between Jan, 10th and May, 1st 2017. The API was configured to retrieve tweets within 25km radius of 4 Portuguese cities (Lisbon, Porto, Braga and Faro). Supported by results in [11], it is assumed that the 1% of the tweets retrieved by the API are statistically representative of the hashtag diversity.

In total, 28615 distinct hashtags were found with 19120 of those only appearing once. Figure 1 is a graphical association between the hashtags retrieved and the cities where they have been published. The histogram on Fig. 2 shows that the majority of tweets (54%) only has 1 hashtag and that 80% of the tweets

[3] https://dev.twitter.com/streaming/overview

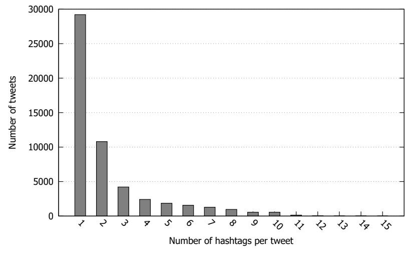

Fig. 2: Hashtags per tweet

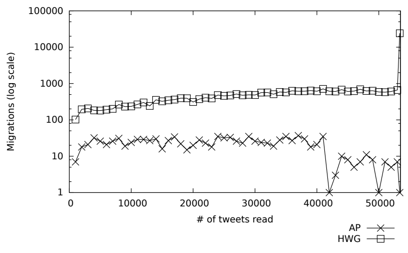

Fig. 3: State items migrations

have 3 or less hashtags. Overall, most of the tweets (66%) originated in Lisbon with Porto, Braga and Faro following with respectively 22%, 8% and 4%.

The evaluation consisted in running the algorithms described in Sec. 4 over an application state created from the tweets set. Each tweet was mapped on a transaction, with its hashtags representing the state items accessed. The transaction is assumed to have occurred on the location where the tweet was published and this location is associated to 1 of a set of 4 possible surrogates mapped on the cities. Each hashtag is initially affected to the location where it was first tweeted. Transactions are processed in chronological order.

5.1 Metrics

The performance of each policy was evaluated using 3 distinct metrics. The **number of state item migrations** and the **number of surrogates participating in a transaction** reflect the overhead imposed to the system. They are expected to be as small as possible in order to increase performance and reduce overhead. The **proportion of transactions enclosed in a single surrogate** complements the metrics above by highlighting the cases where the algorithm was able to achieve the optimal goal of deploying all the items in the same surrogate prior to a transaction.

5.2 Results

State Item Migrations Migrations are time and resource consuming operations, requiring the participation of two surrogates and possibly halting ongoing transactions. The goal of the algorithms is to be as efficient as possible by keeping this value low. The amount of group changes performed in the system is depicted in Fig. 3. The figure omits the Static and MtN algorithms. The former because no migrations are performed and the second because surrogate migrations are necessarily performed prior to each transaction and therefore results cannot be aggregated in the 1000 transaction slices.

The figure 3 shows a clear distinction in the performance of the AP and HWG algorithms. AP is considerably more conservative in what respects to items migration, approaching optimal results as time progresses. On average, AP achieves an interesting result of 0.01 migrations per transaction. This result tends to indicate that once consolidated in a location, hashtags tend to be consistently more used in that location than in the remaining.

In contrast, HWG starts with an amount of group changes close to 0.1 per transaction and slowly increases during the experiment to around 1 group change per transaction. Results suggest that the creation of new edges or their update with new weights has a major impact on the graph partitioning algorithm, creating a map significantly different from the one defined in the previous revision. Part of this problem is attributed to the lack of memory of the algorithm, which focus exclusively on the actual state, disregarding the previous deployment. In practice this leads for example to the random breaking of ties, something that considering the overhead involved in state migration should be avoided.

Figure 3 highlights two abnormalities. The first happens at the 42000 tweets, after which the performance of the AP algorithm improves consistently. A careful analysis of the dataset revealed that 942 of the 1000 transactions in this period contain at least one of the hashtags *BBMAs* and *BTSBBMAs*. Such a large proportion clearly reduces the number of migrations to be performed as the number of tweets using the remaining hashtags was negligible. This trend continued until the end of the dataset, thus benefiting AP's memory approach. Interestingly, the behavior of HWG is orthogonal to this abnormally has it continued to shuffle hashtags unused in the period to leverage partitions. In contrast, the second abnormally, which occurred at the end of the dataset has a strong impact on HWG and can be explained by the small size of the evaluation window (392 transactions). It causes the graph to suffer from a temporary imbalance leading to migrations that could have been avoided with a larger and steadier observation window and confirms the inadequacy of high deployment revision frequencies.

Surrogates per Transaction Figure 4 depicts the average number of surrogates participating on each transaction. The plots of MtN and Static serve as control variables as they represent the theoretical minimum (recall from Sec. 4.1 that MtN moves all items to the same surrogate before every transaction) and the cost of not reacting to changes in utilization patterns. An higher average signals an undesirable situation negatively impacting performance on more transactions.

In the general case, HWG and AP show results between the two control algorithms, confirming that it is possible to improve the performance in scenarios like the one anticipated by our system model. HWG consistently exhibits an average below AP. This result is inline with our expectations, considering the additional effort performed by HWG during the revision deployment phase and which is clearly contributing for an improved geo-aware deployment.

Not surprisingly, the abnormality around the 42000 transactions discussed above can be equally observed in this metric, with all algorithms exhibiting a sudden spike in that region of the plot. The strong concentration of tweets using two particular hashtags is even sufficient for HWG and AP to surpass the average presented by the Static baseline test. To understand this behavior, it should be noticed that the hashtags must be deployed in one of the surrogates

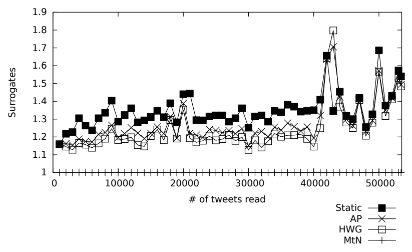

Fig. 4: Surrogates per transaction

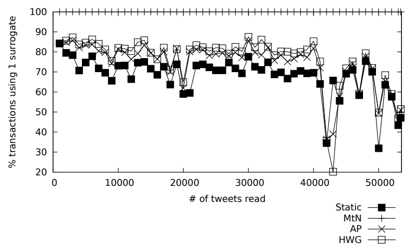

Fig. 5: Transactions enclosed in one surrogate

while a considerable proportion of the tweets are being requested by the remaining. These behaviors can only be mitigated by including in the experiments the datacenter meta-surrogate discussed in the Sec. 2 which would absorb these patterns by claiming the ownership of this hashtags. However, it should be noticed that both algorithms resume their performance once the exceptional scenario has been surpassed.

Single Surrogate Transactions The proportion of transaction that have all their items hosted by the same surrogate is depicted in Fig. 5. Expectations are that the geo-aware state deployment policies approach the perfect scenario presented by design of the MtN control algorithm.

Results confirm that geo-aware deployment policies are all able to outperform the results of the Static baseline test, with a similar gain of around 10% transactions. On average, HWG outperforms AP by approximately 1%. Not surprisingly, the proportion of transactions enclosed in a single group is also considerably hampered at the 42000-43000 transaction mark for the reasons discussed above.

Discussion A preliminary surprising result of the evaluation above was the good performance exhibited by the Static algorithm, capable of competing with other more informed approaches. However, this is a result whose applicability is limited as it strongly depends of the state utilization pattern. In particular, the results of Static show that in a fair proportion of the cases, tweeter hashtags are influenced by locality. The aggressive MtN algorithm equally presents interesting results. However, it cannot be directly compared to the remaining as their deployment revision frequencies are significantly different.

Overall, the performance of any geo-aware state deployment algorithm will be dictated by the trade-off between migrating state items between surrogates and performing distributed transactions, i.e., those involving more than one surrogate. Figures 6 and 7 materialize this trade-off in the results observed with the tweeter dataset from two distinct perspectives.

Figure 6 depicts a two-dimension perspective of the absolute number of item migrations against the absolute number of local transactions observed in our

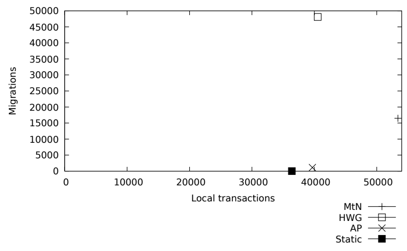

Fig. 6: Transactions in one surrogate per change

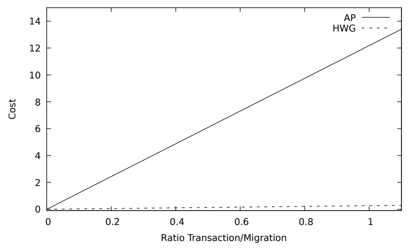

Fig. 7: Relation between algorithms

tweeter dataset. Best algorithms are expected to be concentrated at the bottom right corner of the plot, thus presenting the maximum number of local transactions using the minimal number of item migrations. Not surprisingly, champions on each dimension are the Static and MtN algorithms. However, as discussed before, they do not comply with the expected behavior.

The comparison between AP and HWG clearly suggests an advantage for AP. However, this is exclusively due to the excessively large number of item migrations required by the graph partitioning algorithm as HWG surpasses AP in the number of local transactions. A research direction that is clearly highlighted by this paper is the need to identify the constraints at the partition algorithm that refrain the number of migrations. In this respect, some form of "memory" that permits to preserve the association of items to a surrogate will be one of the topics to pursue.

In a real setting, the differences between these or other algorithms will necessarily be influenced by the effective cost of two operations: migrate a state item and the overhead of performing a distributed transaction instead of a local one. The materialization of these costs is outside the scope of this paper. However, Fig. 7 represents the overall "penalty" of AP and HWG for our experiment as a function of the ratio between the two costs. The plots represent the equation $\text{overhead}_a(x) = x(53392 - L_a) + \frac{1}{x}M_a$ where L_a and M_a are respectively the number of local transactions and migrations for algorithm a and x is the cost ratio between migrations and distributed transactions.

Again the figure highlights a clear disadvantage for HWG, with its overhead only being lower than AP when the ratio is very close to 0, i.e. when migrations have a negligible cost in comparison with distributed transactions. However, it should be noted that these results may be negatively influenced by the small proportion of transactions involving two or more state items in this dataset, as shown in Fig. 2.

6 Conclusion and Future Work

This paper have proposed and compared a new graph-based algorithm for the geo-aware state deployment problem. In contrast with the alternatives, the graph-based algorithm is "transaction aware" in the sense that it considers associations frequently observed between state items and attempts to keep them in proximity. In spite of its benefits, the advantages of this algorithm can only be observed in a very limited number of scenarios.

Work in this field continues. In particular, the current graph representation is oblivious to important factors like the benefits of keeping each item in the same location, avoiding migrations or long term memory eviction. Finally, authors plan to experiment these algorithms on distinct datasets as an approach to confirm the results observed.

Acknowledgments

Work described in this paper was partially supported by Fundação para a Ciência e Tecnologia, Portugal, under project PTDC/EEI–ESS/5863/2014 - doit.

References

1. Bonomi, F., Milito, R., Natarajan, P., Zhu, J. In: Fog Computing: A Platform for Internet of Things and Analytics. Springer Int'l Publishing, Cham (2014) 169–186
2. Curino, C., Jones, E., Zhang, Y., Madden, S.: Schism: A workload-driven approach to database replication and partitioning. Proc. VLDB Endow. **3**(1-2) (2010) 48–57
3. Elnikety, S., Dropsho, S., Zwaenepoel, W.: Tashkent+: Memory-aware load balancing and update filtering in replicated databases. In: Proc. of the 2Nd ACM SIGOPS/EuroSys European Conf. on Computer Systems 2007. (2007) 399–412
4. Pai, V.S., Aron, M., Banga, G., Svendsen, M., Druschel, P., Zwaenepoel, W., Nahum, E.: Locality-aware request distribution in cluster-based network servers. In: Proc. of the 8th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. ASPLOS VIII, ACM (1998) 205–216
5. Pavlo, A., Curino, C., Zdonik, S.: Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems. In: Proc. of the 2012 ACM SIGMOD Int'l Conference on Management of Data. SIGMOD '12, ACM (2012) 61–72
6. Asad, O., Kemme, B.: Adaptcache: Adaptive data partitioning and migration for distributed object caches. In: Proceedings of the 17th International Middleware Conference. Middleware '16, New York, NY, USA, ACM (2016) 7:1–7:13
7. Wu, Z., Butkiewicz, M., Perkins, D., Katz-Bassett, E., Madhyastha, H.V.: Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. SOSP '13, New York, NY, USA, ACM (2013) 292–308
8. Hoang, L.L., Bezerra, C.E., Pedone, F.: Dynamic scalable state machine replication. In: 46th IEEE/IFIP Int'l Conf. Dependable Systems and Networks. (2016)
9. Sciascia, D., Pedone, F.: Geo-replicated storage with scalable deferred update replication. In: Proc. of the 2014 IEEE 33rd Int'l Symposium on Reliable Distributed Systems Workshops. SRDSW '14, IEEE Computer Society (2014) 26–29
10. Karypis, G., Kumar, V.: Multilevelk-way partitioning scheme for irregular graphs. J. Parallel Distrib. Comput. **48**(1) (January 1998) 96–129
11. Morstatter, F., Pfeffer, J., Liu, H., Carley, K.M.: Is the sample good enough? comparing data from twitter's streaming API with twitter's firehose. CoRR **abs/1306.5204** (2013)