The Multi-Tenant Data Placement Problem

Jan Schaffner*, Dean Jacobs†, Tim Kraska‡, and Hasso Plattner*
*Hasso-Plattner-Institute
University of Potsdam
August-Bebel-Str. 88
14482 Potsdam, Germany
Email: firstname.lastname@hpi.uni-potsdam.de
†SAP AG
Dietmar-Hopp-Allee 16
69190 Walldorf, Germany
Email: dean.jacobs@sap.com
‡UC Berkeley
465 Soda Hall MS-1776
Berkeley, CA 94720, USA
Email: kraska@berkeley.edu

***Abstract*—With the advent of the Software-as-a-Service (SaaS) deployment model, managing operational costs becomes more and more important for providers of hosted software. The cost for hosting and providing a service is directly proportional to the operational margin that can be achieved when running a SaaS business. Possible avenues for reducing operational costs are consolidation (i.e. co-locating multiple customers onto the same server) and automation of cluster management (i.e. migration of customers between servers, automatic replication for performance or high availability). In this paper, we propose the formalization for the problem of assigning "tenants" (i.e. the customers) to servers of an on-demand database cluster. We will pose this problem in the form of an optimization problem, omitting database specifics and thus presenting the problem in an abstract fashion, using the metaphor of assigning tokens to baskets (i.e., tenants to servers). This formalization is both a first step towards solving the placement problem in an efficient way and a helpful basis for comparing multiple solutions to the problem to each other.**

***Index Terms*—multi-tenancy; software-as-a-service; databases; enterprise applications; column-store**

I. Introduction

Implementations of Enterprise SaaS commonly maintain data in a farm of conventional databases. To reduce total cost of ownership, multiple tenants are consolidated into each database instance [1]. As an example, in October 2007, the SaaS CRM vendor RightNow had 3,000 tenants distributed across 200 MySQL database instances with 1 to 100 tenants per instance [2]. Table I shows the cost of hosting such a system with and without multi-tenancy on Amazon's Elastic Compute Cloud (EC2) for a year. This calculation assumes each database is run on one small EC2 instance, which may not be sufficient in some cases. The standard on-demand pricing for one such unit is $0.085 per hour. The one-year reserved pricing is $227.50 up front plus $0.03 per hour [3]. These figures do not include the costs to administer the databases, which would further skew the results in favor of multi-tenancy by a wide margin. The moral of the story is clear: the more consolidation the better.

	Single-tenant	Multi-tenant
Standard on-demand pricing	$2,233,800	$148,920
One-year reserved pricing	$1,470,900	$98,060

TABLE I
Yearly Cost to Host RightNow Databases on Amazon EC2

SaaS implementations commonly use pre-existing database replication mechanisms for availability and performance. This practice treats each group of tenants as a single unit for the purposes of replication. As a result, the excess capacity that is reserved to handle failures and workload surges is pooled across only a small number of servers and significant over-provisioning is required. In particular, common practice is to maintain master/slave pairs in which each slave gets no traffic so that it can handle the full load in case its master fails. This technique is often referred to as *mirroring* (cf. Figure 1). The downside of mirroring is that in case of a failure all excess workload is re-directed to the other mirror. In doing so, the mirror server is a local hot-spot in the cluster until the failed server is back online. A technique for avoiding such hot-spots is to use *interleaving*, which was first introduced in Teradata [4]. Interleaving entails performing replication on the granularity of the individual tenants rather than all tenants inside a database process. This allows for spreading out the excess workload in case of a server failure across multiple machines in the cluster. As a result, excess capacity is pooled across a larger number of servers, the work on any one server is smoothed out with high probability, and less over-provisioning is required. As an example, consider a scenario in which the layout is "perfect" in the sense that, for any two servers, there is at most one tenant with a replica on both servers. If there are 10 tenants per server, then over-provisioning by 50% would allow a failure of up to 5 servers

 ISBN: 978-1-61208-185-4

that share a tenant with any given server. The benefits of this scheme are proportional to the amount of consolidation: with 100 tenants per database, allowing a failure of 5 servers would require over-provisioning by only 5%.

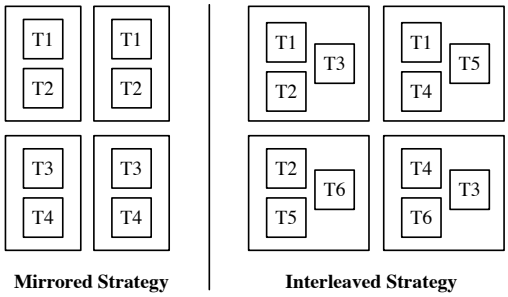

Fig. 1. Example Layouts of Tenant Data

Replica placement is an on-line problem: the layout must be modified as tenants join and leave the system and their resource consumption varies with seasons and success. Placement algorithms must be judged not only on the quality of the layout, but also on the amount of data that must be migrated to achieve it. Migration should be minimized because it consumes resources and it can complicate failure recovery. Web-scale databases generally sacrifice some layout quality in order to avoid excessive migration. For traditional data warehouses, mostly large and expensive server and storage systems are used. For small- and medium-sized companies, it is often too expensive to implement and run such systems. Given this situation, the SaaS model comes in handy, since these companies might opt to run their OLAP at an external service provider. The challenge is then for the analytics service provider to minimize total cost of ownership by consolidating as many tenants onto as few computing resources as possible, a technique often referred to as multi-tenancy [1].

The *Rock* project at the HPI [5], [6], [7], [8] seeks to maximize throughput in a cluster of main memory column databases. The goal is to support the highest possible number of concurrently active users while guaranteeing hard service level objectives on end-user response times (e.g. “99 percent of all queries have to complete in less than 1 second”). We explore different data placement strategies for deciding which tenants are co-located on which servers in order to minimize the number of servers when running a given number of users. This problem is at the heart of large-scale Internet services trying to minimize the cost of their data centers.

This paper is structured as follows: Section II will formulate the placement problem as an optimization problem. We will omit database specifics and pose the problem in an abstract fashion, using the metaphor of assigning tokens to baskets (i.e., tenants to servers). Section III discusses the computational complexity of the placement problem. In Section IV, we discuss related work. Section V concludes this paper.

II. Problem Statement

Let $N = \{i_1, \ldots, i_{|N|}\}$ be a set of baskets and $T = \{t_1, \ldots, t_{|T|}\}$ a set of tokens. An assignment of tokens to baskets is given, as shown in Figure 2. All tokens have a radius $r(t)$ and a color $c(t)$, which are also known for all tokens. Each color occurs exactly twice (or, in other words, each token occurs exactly twice, which means that there are $2|T|$ tokens in an assignment). The problem to be solved is to find a new assignment of tokens to baskets with the following properties:

- No color occurs twice in the same basket.
- The sum of all token radiuses in each basket does not exceed a fixed upper bound $cap(i)$.
- The sum of all token radiuses should have a similar value for all baskets.
- Any two colors appearing together in one basket should preferably not appear together in a second basket.

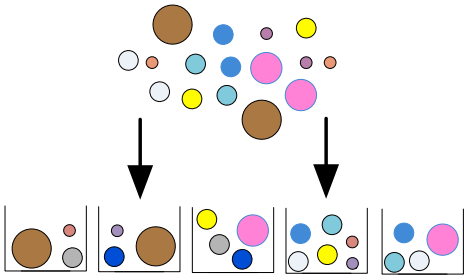

Fig. 2. Assignment of tokens to baskets

A. Formal Description

To denote the assignment of tokens to baskets we define a decision variable y as follows:

$$y_{t,i}^{(k)} = \begin{cases} 1 & \text{if token } t \text{ is in basket } i \\ 0 & \text{otherwise} \end{cases}$$

$$\text{with } k \in \{0,1\},\ t \in T,\ i \in N$$

The index k identifies the first and the second token of the same color, respectively. An assignment of tokens to baskets Y', the number of baskets N, and the set of tokens T with their respective radiuses and colors is given as an input for this problem. The goal is to devise an algorithm which calculates a new assignment Y (i.e. the transformation $f : Y' \rightarrow Y$).

Token radiuses increase and decrease as time progresses, although they are fixed for any given instance if the problem. The model is that a new instance of the problem is created each time one or more changes in token radius have been observed. We are looking for an algorithm that—when invoked—balances the sum of token sizes across all baskets and, at the same time, tries to minimize color co-appearance across the baskets. An optimal assignment in this respect would be such that no two colors appearing together in one basket appear

 ISBN: 978-1-61208-185-4

together in any other basket at the same time. To do so, the algorithm moves tokens between the baskets.

The sum of all radiuses $r(s)$ of the tokens in a given basket i is defined as

$$R(i) := \sum_{t \in T} \sum_{k=0}^{1} y_{t,i}^{(k)} r(t),\ i \in N.$$

To describe how good (or bad) a given assignment of tokens to baskets is w.r.t. color co-appearance, we introduce a penalty function $P(i)$. It is computed from the perspective of an individual basket and is defined as the sum of all token radiuses of co-appearing tokens in one of the other $N-1$ baskets. Since this value depends on which of the other $N-1$ baskets is chosen as a partner in this binary comparison, $P(i)$ is defined relative to the partner yielding the maximum value:

$$P(i) = \max_{j \in N} \left(\sum_{t \in T} \sum_{k=0}^{1} y_{t,i}^{(k)} y_{t,j}^{(1-k)} r(t) \right)$$

$$\text{with } i, j \in N, i \neq j$$

1) Constraints:

1) A valid assignment Y must contain each color exactly twice.
$$\sum_{i \in N} y_{t,i}^{(0)} + y_{t,i}^{(1)} = 2,\ \forall t \in T$$

2) No basket must contain any two tokens of the same color.
$$y_{t,i}^{(0)} + y_{t,i}^{(1)} \leq 1,\ \forall t \in T,\ \forall i \in N$$

3) The sum of all token sizes in a basket i must be less than or equal to the capacity of the basket.
$$R(i) \leq cap_i,\ \forall i \in N$$

2) Objective Functions:

1) All baskets shall be balanced w.r.t. aggregate token size (in addition to constraint no. 3, which only specifies an upper bound for the sum of all token radiuses within one basket R_i). One way of progressing towards a similar value for the different $R(i)$s is to minimize their variance:
$$\min\ Var(R(1), \ldots, R(|N|))$$

2) Co-appearances of colors in the baskets shall be minimized:
$$\min\ \sum_{i \in N} P(i),\ \forall i \in N$$

3) In addition to minimizing the co-appearance penalty per basket (the previous optimization goal), all baskets should have a similar penalty. One way of progressing towards a similar value for the different $P(i)$s is to minimize their variance:
$$\min\ Var(P(1), \ldots, P(|N|))$$

B. Possible Extensions

The general formulation of the problem as posed above can be extended to make it even harder to devise good solutions.

1) Varying number of baskets: For the problem stated above we assume a fixed number of baskets N. It might be the case that the observed changes in token size create a situation in which the current number of baskets is not sufficient for finding an assignment of tokens to baskets such that none of the above constraints is violated. Given such a situation, the algorithm is allowed to create a new basket. Similarly, when the changes in token size result in a situation where all of the above constraints could be satisfied using fewer baskets, then the algorithm can empty a basket by migrating its tokens to other baskets and delete the basket. It would also be conceivable to trade-off the the number of baskets against the balancing of the $R(i)$s as well as the values and the balancing of the $P(i)$s.

2) Minimizing the number of migrations: So far we have not imposed a limit on the number of movements of tokens between baskets necessary to provide the transformation $f : Y' \rightarrow Y$. However, it would be conceivable to try to minimize the number of movements in this sequence. It would also be interesting to study how fast the other goal functions converge to an optimal value, varying the number of allowed migrations in f.

III. Computational Complexity

The tenant placement problem presented above is structurally similar to the general *bin packing with conflicts* (BPC) problem. In addition to standard bin packing with a given set V of n items, an instance of BPC also contains conflict graph $G = (V, E)$, where an edge $(i, j) \in E$ exists if items i and j are in conflict and may thus not be assigned to the same bin. The BPC problem is known to be NP-complete [9]. In order to proof that the tenant placement problem is also NP-complete, it must be shown that:

1) a candidate solution can be verified in polynomial time.
2) an instance of the BPC problem can be reduced to an instance of the tenant placement problem.

1 follows trivially from the observation that all constraints of the tenant placement problem are terms of quadratic or lower complexity. A formal proof of 2 is beyond the scope of this paper. However, the fact that the tenant placement problem only contains additional constraints in comparison to BPC suggests that a reduction is possible. In the remainder of this section we will elaborate on the combinatorial complexity of the tenant placement problem.

One possible way of enumerating all possibilities of assigning the two copies of a tenant to N servers will now be briefly discussed. Given that both copies of a tenant must be on different servers and that both copies are equivalent, all possibilities to assign a single tenant to two servers can be done by creating a matrix with dimensions $N \times N$. The elements of this matrix are defined as follows:

$$(i, j) = \begin{cases} 1 & \text{if } i < j \\ 0 & \text{otherwise} \end{cases},\ i, j \in N$$

 ISBN: 978-1-61208-185-4

# Servers	# Tenants	Search Space
3	16	43,046,721
3	17	129,140,163
4	13	13,060,694,016
3	17	16,926,659,444,736

TABLE II
NUMBER OF COMBINATORIAL POSSIBILITIES FOR SMALL PROBLEM INSTANCES

An example for $N = 4$ servers:

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

The number of possibilities for assigning two copies of a tenant to N servers is thus $N(N-1)/2$. We refer to all 1-valued elements of this matrix as possible *configurations* of a tenant.

We developed a brute force solver for enumerating all possibilities to assign T tenants to N servers. As we have already established above, there are $N(N-1)/2$ possibilities to assign two copies of one tenant to N servers. The number of combinatorial possibilities to assign T tenants to N servers is thus:

$$\left(\frac{N(N-1)}{2}\right)^T$$

In our implementation, we encode each combination using a sequence of length T with each element of the sequence being an integer encoding a tenants *configuration*:

$$(a_0, \ldots, a_{T-1}),\ 0 \leq a_i \leq \frac{N(N-1)}{2} - 1$$

This sequence obeys lexicographic ordering in the sense that the next combination can by computed simply by incrementing the rightmost element of the sequence which is smaller than $(N(N-1)/2) - 1$. Brute force enumeration can nicely be parallelized: In our implementation, we use recursion to generate multiple ranges of the sequence with equal sizes. Each of those ranges is then enumerated in parallel. Our implementation uses Scala actors [10] and fits well with the large number of available cores in modern server machines. Nevertheless, a solution to the placement problem can only be found for very small instances of the problem using brute force. Table II show the steep increase of the search space size for four small instances.

IV. RELATED WORK

The question of how to distribute data in a cluster of databases has a fundamental impact on the overall performance of the cluster. The data placement problem has been systematically studied in the context of parallel databases and, more recently, distributed databases for Web applications.

A. Parallel Databases

In the parallel databases Teradata [11], Gamma [12], Bubba [13] and Tandem [14], the data placement problem entails distributing a fixed collection of relations across a fixed cluster of servers so as to minimize response times. Large relations are fully partitioned, also called *fully de-clustered*, across all servers and thus placement is straight-forward. For small relations, the placement problem is NP-hard [15] and various heuristics have been proposed. In Bubba, small relations are placed in decreasing order of their access frequency, or "heat". At each placement step, the algorithm tries to balance the overall heat at each server. In a simulation study, [16] compares the Bubba algorithm with simple round robin placement of small relations. The conclusion is that, for large numbers of small relations, round robin performs as well as the Bubba algorithm. Round robin is in general preferable because it does not require knowledge of the workload.

Parallel databases generally maintain two copies of the data to ensure high availability. For small relations, a common approach to data placement is to simply treat each copy as a separate relation, with the additional constraint that the two copies cannot be placed on the same server. Bubba, for example, maintains the copies in different formats and tracks the heat of each copy independently.

For large relations, Teradata uses a technique called interleaved de-clustering. Each of the N servers in a cluster is made responsible for the primary copy of one fragment of a relation. The secondary copy of each fragment is divided into $N-1$ sub-fragments that are distributed across the other servers in the cluster. Thus, when a server fails and the primary copy of a fragment becomes unavailable, the work is redistributed across $N-1$ other servers.

A disadvantage of interleaved de-clustering is that, if two servers in a cluster become simultaneously unavailable, one sub-fragment will have no active copy and the entire system has to be shut down. This problem gets worse as the cluster gets bigger, since there are more servers that can fail. As an alternative, Gamma uses a technique called chained declustering [16] in which the servers are organized into a logical ring. Each of the servers is made responsible for the primary copy of one fragment and the secondary is placed immediately after the primary in the ring. If a server fails, its workload is taken over by its successor and predecessor in the ring, which seemingly leads to an unbalanced system. The solution is that the workloads for the other fragments are incrementally shifted around the ring to re-balance the system. To shift the workload for a fragment, the boundary for the range of responsibilities between the two copies is shifted. Chained declustering makes it possible to survive the failure of any two non-adjacent servers in the ring.

In placement problem as presented in this paper, we are solely concerned with the placement of small relations. Interleaving is performed at that level rather than at the level of fragments and sub-fragments of large relations. In the parallel databases, the load on fragments of a large relation

is correlated because every fragment is used to answer a query. The performance characteristics of our system differ because tenants submit queries independently. In our context, data placement is an on-line problem and entails adjusting the size of the cluster so response times requirements are met using as few servers as possible. When the load on a server becomes too high due to increases in the maximum resource consumption of tenants, our placement algorithms make local adjustments to the layout. Parallel databases can accommodate changes in the number and sizes of relations and the number of servers in the cluster, however such changes are infrequent and are applied more globally.

B. Web-Scale Databases

Amazon Dynamo is a distributed key-value store that uses a multi-master-update, lazy-update-propagation model with conflict resolution [17]. Data is distributed across the servers using a variant of *consistent hashing* [18], [19]. The keys are hashed into a fixed circular ring and each server randomly chooses T tokens in the ring. For each of its tokens, a server is responsible for the key range from that token to the next higher token of any server. The replicas for a key range are placed on the next distinct servers moving clockwise around the ring. Thus Dynamo interleaves data: if a server fails, its workload is distributed among T other groups of servers. An essential characteristic of this system is that each key is accessed independently, the keys may be randomly partitioned, and such a partitioning results in a good load distribution for large numbers of keys. In contrast, for Enterprise SaaS and the algorithms studied in this paper, the partitions are fixed at tenant boundaries and the workload varies greatly between tenants.

In Google BigTable [20], rows are range-partitioned across servers by key. Rows with the same key prefix are guaranteed to be adjacent, hence the application can exert control over partitioning. Enterprise SaaS can be implemented on such a system by using the tenant ID as the leading prefix of the key. In BigTable, replication is performed at a lower level by the Google File System [21] and there is never more than one active node for a given data item. Nevertheless, if additional active copies were maintained *using different sort orders for the tenants*, interleaved replication would be provided. The problem with this approach is that the system would unnecessarily maintain the sort order of tenants in each replication group. Our approach is not subject to such a constraint and is not operationally more complex.

C. Data Placement in Other Areas

Problems similar to data placement are also known outside the database community.

The so-called File Allocation Problem (FAP) is concerned with assigning files to nodes. If a file is on a node then access is cheap (local access), otherwise there is some communication cost attached to to accessing the file (remote access). This problem has been extensively studied in literature, and many different models using different objectives for optimization have been proposed. A survey providing a qualitative comparison of these models can be found in [22].

An extension of the FAP is the replica placement problem (RPP) for content delivery networks. As an example, consider an Internet service provider that wants to assign multimedia data to caches in the network topology. Here, the cost of remote accesses is extended by the notion of the distance to the closest node that has a copy of the requested object. [23], [24] surveys and compares known formalizations and solutions to RPP.

All approaches which can be subsumed by either FAP or RPP minimize cost functions which are designed with the aim of improving average data access performance. Utilization and the ability to provide service at a guaranteed performance in the presence of failures is not investigated.

V. Conclusions and Future Work

In this paper, we have given a formal definition of the tenant placement problem, which we have identified as one of the key problems in cluster management for SaaS applications. This formalization is both a first step towards solving the placement problem in an efficient way and a helpful basis for comparing multiple solutions to the problem to each other. We have outlined a mechanism for the brute force enumeration of the complete search space. However, the brute force strategy is only useful for very small instances of the problem. As part of future work we thus plan to work on heuristic methods for solving the placement problem. In doing so, we plan to not only investigate and compare the theoretical properties of multiple placement algorithms but also to implement the algorithms in a multi-tenant database system and experimentally validate their practical usefulness.

Acknowledgements

The authors would like to thank Michael J. Franklin from UC Berkeley for his valuable feedback and comments.

References

[1] D. Jacobs and S. Aulbach, "Ruminations on Multi-Tenant Databases," in *Datenbanksysteme in Business, Technologie und Web (BTW 2007), 12. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), Proceedings, 7.-9. März 2007, Aachen, Germany*, 2007, pp. 514–521.

[2] M. Myer, "Rightnow architecture," in *HPTS*, 2007.

[3] "Amazon ec2 pricing," http://aws.amazon.com/ec2/pricing/, last visited 2011-12-16.

[4] D. J. DeWitt, M. G. Smith, and H. Boral, "A Single-User Performance Evaluation of the Teradata Database Machine," in *High Performance Transaction Systems, 2nd International Workshop, Asilomar Conference Center, Pacific Grove, California, USA, September 28-30, 1987, Proceedings*, 1987, pp. 244–276.

[5] J. Schaffner, B. Eckart, D. Jacobs, C. Schwarz, H. Plattner, and A. Zeier, "Predicting in-memory database performance for automating cluster management tasks," in *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, 2011, pp. 1264–1275.

[6] J. Schaffner, D. Jacobs, B. Eckart, J. Brunnert, and A. Zeier, "Towards enterprise software as a service in the cloud," in *Workshops Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, 2010, pp. 52–59.

 ISBN: 978-1-61208-185-4

[7] J. Schaffner, B. Eckart, C. Schwarz, J. Brunnert, D. Jacobs, A. Zeier, and H. Plattner, "Simulating Multi-Tenant OLAP Database Clusters," in *Datenbanksysteme für Business, Technologie und Web (BTW), 14. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 2.-4.3.2011 in Kaiserslautern, Germany*, 2011, pp. 410–429.

[8] M. Grund, J. Schaffner, J. Krüger, J. Brunnert, and A. Zeier, "The effects of virtualization on main memory systems," in *Proceedings of the Sixth International Workshop on Data Management on New Hardware, DaMoN 2010, Indianapolis, IN, USA, June 7, 2010*, 2010, pp. 41–46.

[9] M. Gendreau, G. Laporte, and F. Semet, "Heuristics and lower bounds for the bin packing problem with conflicts," *Computers & OR*, vol. 31, no. 3, pp. 347–358, 2004.

[10] P. Haller and M. Odersky, "Actors That Unify Threads and Events," in *Coordination Models and Languages, 9th International Conference, COORDINATION 2007, Paphos, Cyprus, June 6-8, 2007, Proceedings*, 2007, pp. 171–190.

[11] D. Flynn and M. Adamson, "Capacity Planning on a Teradata DataWarehouse," in *34th International Computer Measurement Group Conference, December 7-12, 2008, Las Vegas, Nevada, USA, Proceedings*, 2008, pp. 93–104.

[12] D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H.-I. Hsiao, and R. Rasmussen, "The Gamma Database Machine Project," *IEEE Trans. Knowl. Data Eng.*, vol. 2, no. 1, pp. 44–62, 1990.

[13] G. P. Copeland, W. Alexander, E. E. Boughter, and T. W. Keller, "Data Placement In Bubba," in *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, June 1-3, 1988*, 1988, pp. 99–108.

[14] P. Celis, "Distribution, Parallelism, and Availability in NonStop SQL," in *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, June 2-5, 1992*, 1992, p. 225.

[15] D. Saccà and G. Wiederhold, "Database Partitioning in a Cluster of Processors," *ACM Trans. Database Syst.*, vol. 10, no. 1, pp. 29–56, 1985.

[16] H.-I. Hsiao and D. J. DeWitt, "Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines," in *Proceedings of the Sixth International Conference on Data Engineering, February 5-9, 1990, Los Angeles, California, USA*, 1990, pp. 456–465.

[17] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, 2007, pp. 205–220.

[18] 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 *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1997, pp. 654–663.

[19] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, 2003.

[20] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, 2008.

[21] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*, 2003, pp. 29–43.

[22] L. W. Dowdy and D. V. Foster, "Comparative Models of the File Assignment Problem," *ACM Comput. Surv.*, vol. 14, no. 2, pp. 287–313, 1982.

[23] Y. Saito, C. T. Karamanolis, M. Karlsson, and M. Mahalingam, "Taming Aggressive Replication in the Pangaea Wide-Area File System," in *OSDI*, 2002.

[24] M. Karlsson and C. T. Karamanolis, "Choosing Replica Placement Heuristics for Wide-Area Systems," in *24th International Conference on Distributed Computing Systems (ICDCS 2004), 24-26 March 2004, Hachioji, Tokyo, Japan*, 2004, pp. 350–359.

 ISBN: 978-1-61208-185-4