An epidemic approach to dependable key-value substrates

Miguel Matos, Ricardo Vilaça, José Pereira and Rui Oliveira
Universidade do Minho
Braga, Portugal
{*miguelmatos, rmvilaca, jop, rco*}*@di.uminho.pt*

***Abstract*—The sheer volumes of data handled by today's Internet services demand uncompromising scalability from the persistence substrates. Such demands have been successfully addressed by highly decentralized key-value stores invariably governed by a distributed hash table. The availability of these structured overlays rests on the assumption of a moderately stable environment. However, as scale grows with unprecedented numbers of nodes the occurrence of faults and churn becomes the norm rather than the exception, precluding the adoption of rigid control over the network's organization.**

In this position paper we outline the major ideas of a novel architecture designed to handle today's very large scale demand and its inherent dynamism. The approach rests on the well-known reliability and scalability properties of epidemic protocols to minimize the impact of churn. We identify several challenges that such an approach implies and speculate on possible solutions to ensure data availability and adequate access performance.

***Keywords*-Data store, Tuple store, Epidemics, Gossip-based dissemination**

I. Introduction

Cloud Computing is often portrayed as the panacea of all IT needs, promising infinitely scalable services as a commodity. Part of the attractiveness of this perspective can be attributed to the ever growing need of scalable and dependable IT services, either in the form of storage or computing, by enterprises and the general public that tend to increasingly act on a global scale.

Still this general overarching need is far from being satisfied and, in particular, the need for a scalable and dependable data store capable of operating with unprecedented volumes of clients and ever growing data [1], [2] that needs to be stored, retrieved and most importantly processed in order to obtain usable information. This fueled the emergence of a new generation of data stores capable of handling large volumes of data and requests, albeit forfeiting the rich interface and consistency guarantees found in traditional database systems [3], [4], [5], [6].

Architecturally, traditional relational database management systems (RDBMS) offer strong consistency models but are confined to a single node, or rely on distributed coordination protocols that are known to scale only to a few dozen nodes [7]. Novel distributed approaches relax the consistency model in order to scale to hundreds of nodes at the cost of simple few single-item guarantees.

The classical approach to distribution is by recurring to structured substrates, where nodes are organized in a well defined and strictly controlled fashion. This is for instance done in Cassandra [4], by organizing nodes in a Distributed Hash Table. While highly efficient, the rigid structure and organization of DHTs is sensible to faults and churn [8]. Structure maintenance in a dynamic environment is hard because several invariants need to be observed and costly as repair mechanism are reactive and thus induce an overhead proportional to churn.

Unfortunately, as the system size grows, churn becomes pervasive due to hardware and software failures as well as system reconfigurations [9]. For instance, field studies show error rates in RAM as high as 8% [10] and disk replacements rates up to 13% [11]. Furthermore, there is evidence that failure rates grow at least linearly with the system size and are affected by system load [12] both important aspects in existing infrastructures. While those studies present already high failure rates of hardware components, churn is expected to be higher due to transient failures of these and other components. The purported scalability rests therefore on the assumption of a moderately stable environment that tends to vanish as the system grows. Most strikingly, with systems being designed at million node scales [9] to cope with next generation demands at a global scale current designs are lacking: knowing all nodes to perform some operations as in Cassandra is unattainable, as is the requirement of a (set of) master nodes as in Bigtable/HBase [13], [5].

Scenario: The alternative seems thus to rely on a substrate that scales and is able to cope with a churn and faulty environment as a consequence of scale itself. This scenario ranges thus from the typical data center made of commodity hardware but also organizations with a large existing computing infrastructure such as universities and global enterprises, where the common denominator is very large scale and high dynamism.

In this position paper we discuss our current effort on applying an epidemic-based approach to the problem of massive data store systems, presenting the general architecture of the proposed system, its current status and open challenges. The rest of the paper is organized as follows: in Section II we present our two-layered system architecture, describing the requirements of each layer, how they integrate and its current status. Then in Section III we discuss several open

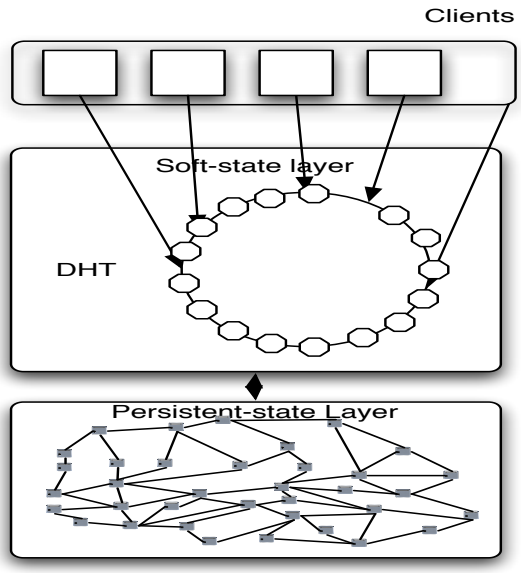

Figure 1. DataDroplets architecture

design questions, the research path we intend to follow and the expected open problems.

II. DataDroplets

The guiding principle behind our proposal is clear separation of concerns between different functional aspects of the system that we addressed at different abstractions levels with different goals and assumptions.

The architecture we propose is driven by a careful analysis of client requirements and expectations that could be broadly divided in two major domains: i) client interface and concurrency control and ii) data storage itself and low level processing. Most strikingly this is the approach taken by traditional relational database management systems (RDBMS) and sidestepped by new data store proposals. As a matter of fact, in RDBMS the client has access to a clearly defined interface that allows the specification of details such as isolation guarantees and primitives for performance tuning such as indexes, but hides the details of how data is actually stored, maintained and processed. This is in sharp contrast with popular data processing approaches such as MapReduce [14].

We address these two distinct domains with the hybrid architecture depicted in Figure 1.

At the top, a soft-state layer is responsible for handling client requests and avoiding issues such as conflicts that eventually arise due to concurrency. Those problems require a careful ordering of requests and thus coordination among participating nodes which is best achieved by a structured DHT-based approach where nodes partition the key-space among themselves in order to achieve load-balancing and unequivocal responsibility for partitions [15]. With requests ordered, the remaining and costly procedure of performing the actual read or write over the data is delegated to the persistent-state layer below. As the soft-state layer only carries simple and lightweight operations over metadata, which can be maintained in memory, we expect it to be moderately sized and thus manageable with a structured approach. In fact, on the event of a catastrophic failure, or when a new node joins this layer, metadata can be reconstructed from the data reliably stored at the underlying persistent-state layer.

The design of the soft-state layer also takes into account several performance-wise considerations. We take advantage of spare capacity to serve as a tuple cache [16] thus avoiding unecessary operations at the persistent-state layer. As the soft-layer always knows the most recent version of an item, cache inconsistency issues are eliminated. This is also leveraged to improve search performance: as the (last) version of the item to retrieve is always well-known the use of quorums at the persistent-state layer is not necessary. Maintaining knowledge of some of the nodes that store the data in the persistent-state layer is also a straightforward technique to improve operation performance [17]. Further, the use of effective tuple placement strategies that take into account, for instance, the correlation among tuples also has a significant impact on performance as studied in [18].

With the major problems of the upper layer mostly addressed we are now shifting our research focus to the underlying persistent-state layer which is where the actual data storage and processing takes place. Due to the scale of both nodes and data the approach is to leverage on the scalability and resilience of epidemic-based protocols. The fundamental requirements of this layer are data availability, operation performance and processing capabilities. The only assumption we do so far is that write operations are correctly ordered by the soft-state layer.

III. Proposal and Open Problems

In this section we describe the main design ideas to the epidemic-based persistent state layer, discuss open problems and set the roadmap for future research. The essential properties of a low level storage system are i) data availability and ii) performance: without the first the system is unusable, the second makes it desirable.

In an initial approach we assume simple read and write operations. They are ordered and identified with a request version. Any node in the system may receive requests to perform such operations.

The key idea is to rely on an epidemic dissemination protocol to spread data and operations to relevant nodes, taking advantage of the inherent scalability and ability to mask transient node and link failures. We first discuss the problem of correctness, data availability and durability, and

then how the system can be improved to achieve good performance.

A. Data availability and durability

Data availability is mainly affected by failures and churn as data present in offline nodes becomes inaccessible. The only way to cope with unavailability due to offline nodes is by redundancy and thus the main concern about that availability is how to achieve and maintain adequate redundancy levels, which, despite the redundancy strategy used boils down to the question: how many nodes need to replicate an item.

Upon a write request, it is necessary to ensure that several copies of the item are spread throughout the system for durability. Due to the low capacity of individual nodes and high churn rates it is impossible to track down the state of individual nodes and do globally informed data placement decisions. Our strategy is instead based on a global dissemination/local decision approach.

The key idea is to spread data in an epidemic fashion [19], [20], [21] and have nodes locally decide if they need to store that data. The cost, in terms of network and processor usage, of achieving atomic infection, i.e. reaching all nodes in the system is however considerable as nodes need to relay messages to $ln(N) + c$ neighbors, where N is the system size and c a parameter related to the probability of atomic infection, given by $p_{atomic} = \epsilon^{-\epsilon^{-c}}$. Thus supposing a system with $50\ 000$ nodes, in order to achieve atomic infection with high probability ($p_{atomic} = 0.999 \rightarrow c = 7$) each node will have to relay around 18 copies of each single message ($ln(50\ 000) + 7 \approx 18$).

Unfortunately, atomic dissemination is not *sufficient* to ensure data availability due to the presence of faults and churn. This is because even if updates reach all nodes, it is not guaranteed that the node responsible for keeping that update is available. Redundancy is thus necessary to mask faults and churn and ensure data availability. Most strikingly, with an uniform redundancy strategy (i.e. with copies evenly distributed throughout the system) atomic dissemination is not even *necessary* as it is enough to reach a proportion of the system that covers the required number of replicas. This allows the system to have relaxed atomic dissemination guarantees which directly impact the overhead of dissemination. In fact going from reaching a major portion of the population to guaranteeing atomic dissemination requires a substantial increase in the number of the number of copies that need to be relayed. The combination of the replication factor with the dissemination effort yields an interesting trade-off that requires further exploration.

The remaining challenge is then to decide which nodes should keep the data, restricted by the impossibility to store all data in a single node and by the required redundancy level. Our idea is to address this by means of local sieves that should retain only small fractions of data. Thus upon reception of a new message, nodes locally decide if the message falls into the sieve range and relay it to fanout neighbors. This is in fact similar to what is done in structured DHT approaches where each node is responsible for a given portion of the key space [15], [22]. The sieve function can be computed locally in a random fashion or take into account some similarity metric, either computed by the node itself or as hinted by the soft-state layer. The only correctness requirement is that all the possibilities in the key space are covered in order to avoid data-loss. This gives also enough flexibility to cope with nodes with disparate storage capabilities, as it is only a matter of adjusting the sieve grain in order to impact the amount of stored data. A simple sieve function could simple store locally an item with probability given by $1/number\ of\ nodes$. The number of nodes could be estimated also in an epidemic manner as in [23]. Using replication, the sieve function could be simply extended to take into account the replication degree, r, as $r/number\ of\ nodes$.

The other issue to address is the maintenance of the redundancy levels, accordingly to the redundancy strategy chosen, which should ensure that a minimal number of copies of every item exist in the system. Again due to scale and churn a centralized deterministic approach is infeasible and thus we must rely on probabilistic epidemic-based methods. Those methods, based on random walks [24], [25], allow each node to obtain an uniform sample of the data stored at other nodes and eventually determine how many copies of the items it holds exist in the system. Doing this on a tuple level is however clearly impractical, as it will require a random walk per tuple and very long walks in order to obtain a good estimation with high probability. However, as tuples are retained at nodes according to the sieve function, obtaining an estimate of how many nodes have a given sieve (which corresponds being responsible for a given portion of the key space) suffices. This drastically reduces random walk length and the number of random walks needed as many tuples may be checked at once. To further improve the reliability of the redundancy management it is further possible to have nodes responsible to the same key space (discovered by the random walk procedure) check tuple redundancy directly between them and restore redundancy as necessary.

Due to the dynamic nature of the target scenario further considerations may be taken into account. Churn is expected to be much more dramatic than failures as it is more likely that nodes suffer from transient faults solved with a reboot than from permanent failures that imply a definite departure from the system [11]. This means that redundancy constrains can be relaxed as the vast majority of nodes are expected to recover within a small time window. Another aspect to take into account in the redundancy policy is that the individual capacity of nodes is despicable when compared to the total volume of data which makes the effect of churn and failure

of a single node negligible. Nonetheless, a mechanism to maintain redundancy at acceptable levels is essential to avoid data loss.

B. Data Performance

Whereas data availability is mainly concerned with *how many* nodes need to hold replicas of a given item, data performance is related to *where* those items actually are. There are several strategies to improve performance in such a system, here we consider the following: i) collocation of related items and ii) ordering of items.

1) Item collocation: The most straightforward approach to item co-collocation is by using smarter sieve functions that, instead of blindingly keep items based on a key, are able to take advantage of tuple correlation and thus locally co-locate related items. In fact the use of this strategy at the soft-state layer already showed that performance can be significantly improved when tuple correlation is taken into account [18]. Thus, the soft-state layer can provide hints on which sieve functions should be used and share similar performance-wise advantages. An open challenge with this approach is how to properly disseminate those custom sieve functions ensuring full coverage of the key space and adequate load-balancing.

Another interesting approach to explore is to rely exclusively on the inherent data distribution of items through some value domain. In fact, knowing that the stored data follows a given distribution enables the construction of effective sieves that achieve both precise item collocation and load balancing. For instance, if data follows a normal distribution, sieves located near the mean $\pm$ standard deviation need to be much finer than sieves outside that region due to the higher item density. Such a simple approach enables a wide range of performance trade-offs, for instance to load balance according to disk space or item request popularity simply by choosing the appropriate metric to generate the distribution, and thus avoid hotspots. The challenge is then how to obtain an adequate estimation of item distribution according to some criteria in a decentralized and scalable fashion. Recent work on this subject [26], [27] based itself on epidemic techniques show that it is possible to obtain accurate estimation of distribution for a given parameter in a scalable and lightweight fashion, which can be used as initial building blocks to our approach. Still, our scenario has particular characteristics that may affect the effectiveness of those protocols, namely a large number of duplicates [27] due to the redundancy, and high churn rates [26], [28] due to large scale, which require further investigation.

2) Item ordering: The other essential issue to improve performance is the ordering of items which would enable efficient range scans of items and the construction of advanced abstractions such as indexes, an essential performance feature of traditional relational database management systems that is still lacking on new generation data stores. This problem becomes more evident when data needs to be drawn from several nodes which is, at first, beneficial due to the ability to perform parallel reads but raises several interesting challenges as node organization is essential to obtain adequate performance.

To attain such ordering, as nodes cannot store locally all the data, the natural approach is to order nodes such that each node knows the *next* node from which data needs to be retrieved/processed. In an overlay network this reduces to establishing the appropriate neighbor relations among nodes taking into account the values they store. This semantic organization of nodes clearly calls to an approach based on content-based systems which are precisely suited to arrange nodes taking into account the values of items they hold[29], [30], [31]. While attractive this rigid approach may not be suitable to an environment subject to churn as a precise organization of nodes is not feasible. On the other hand the approach to deal with item collocation by generating item distribution seems remarkably suited to serve as a basis for item (node) ordering. Assuming that nodes locally possess an estimation of the data distribution and as they know the position of the locally stored items in such distribution it is straightforward to assess how *close* in the data space their actual neighbors are. With this knowledge it is possible to establish a partial order among nodes and have them converge to the proper neighborhood using well-known methods [32].

Naturally, such an organization only makes sense from the point of view of a single attribute/value. Thus it is necessary to support several contending such organizations in order to offer range scans and indexes on several attributes. A first naive approach could be to maintain several independent overlays to support distinct ordering but this is not scalable as it imposes an high overhead that grows linearly with the number of nodes. This can be addressed by more complex approaches that are able to cope with such contending organizations in a scalable, albeit sensitive to churn, fashion [33]. Alternatively, recent work was shown that it is possible to support several independent such organizations in an efficient and scalable fashion without ever compromising the resilience of the underlying protocol to faults and churn [34].

C. Data Processing

Another active area of research is the offering of processing capabilities in the new generation data stores [35], [36]. This processing increases the applicability of existing systems by enabling the extraction of structured knowledge from existing data ranging from simple summaries such as counts or maximums, to more complex and elaborate relational joins providing table materialization.

Interestingly, due to the simple composed approach proposed so far, it is straightforward to offer simple aggregations to clients with minimal overhead. In fact, basic distributed computations are already done in order to estimate

the data distribution of a given parameter, and thus it is simply a matter of exposing such results to the soft-state layer. This is attractive as those computations are likely to be required for attributes where an index or range scan is already built and thus can be obtained at no cost. For other attributes a first approach can consist of relying on existing aggregation protocols [37] to continuous compute a given result over the data. Nonetheless some of the challenges pointed above, such as robust aggregation within the dynamic environment and how to cope with multiple instances of data due to redundancy still remain.

The most interesting challenge seems to be to offer relational properties based on a join operator. Effective joins in such an environment are known to be hard to achieve due to the distributed and dynamic nature of the system but at the same time of major relevance to offer functionality closer to the relational model.

ACKNOWLEDGEMENTS

Partially funded by the Portuguese Science Foundation (FCT) under project Stratus: A Layered Approach to Data Management in the Cloud (PTDC/EIA-CCO/115570/2009) and grants SFRH/BD/38529/2007 and SFRH/BD/62380/2009.

We would like to thank the anonymous reviews for their helpful comments in improving the quality of the paper.

REFERENCES

[1] J. Gantz, "The Expanding Digital Universe," IDC White Paper - sponsored by EMC, Tech. Rep., 2007. [Online]. Available: http://www.emc.com/collateral/analyst-reports/expanding-digital-idc-white-paper.pdf

[2] ——, "The Diverse and Exploding Digital Universe," IDC White Paper - sponsored by EMC, Tech. Rep., 2008. [Online]. Available: http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf

[3] 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 *Symposium on Operating Systems Principles*, 2007, pp. 205–220.

[4] A. Lakshman and P. Malik, "Cassandra - A Decentralized Structured Storage System," in *Workshop on Large Scale Distributed Systems and Middleware*, 2009.

[5] The Apache Software Foundation, "HBase," 2011. [Online]. Available: http://hbase.apache.org/

[6] B. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "PNUTS: Yahoo!'s hosted data serving platform," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1277–1288, 2008.

[7] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," *SIGMOD Rec.*, vol. 25, pp. 173–182, June 1996. [Online]. Available: http://doi.acm.org/10.1145/235968.233330

[8] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a dht," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ser. ATEC '04. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10. [Online]. Available: http://portal.acm.org/citation.cfm?id=1247415.1247425

[9] J. Dean, "Designs, Lessons and Advice from Building Large Distributed Systems," in *International Workshop on Large Scale Distributed Systems and Middleware*, 2009, p. Keynote Speech. [Online]. Available: http://www.cs.cornell.edu/projects/ladis2009/talks/dean-keynote-ladis2009.pdf

[10] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild," in *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems - SIGMETRICS '09*. New York, New York, USA: ACM Press, Jun. 2009, p. 193. [Online]. Available: http://portal.acm.org/citation.cfm?id=1555349.1555372

[11] B. Schroeder and G. Gibson, "Disk failures in the real world : What does an MTTF of 1 , 000 , 000 hours mean to you ?" *Usenix Conference on File and Storage Technologies*, pp. 1–16, 2007.

[12] ——, "A large-scale study of failures in high-performance computing systems," in *International Conference on Dependable Systems and Networks (DSN'06)*. IEEE, Jun. 2006, pp. 249–258. [Online]. Available: http://portal.acm.org/citation.cfm?id=1135532.1135705

[13] F. Chang, J. Dean, S. Ghemawat, D. Hsieh, W.and Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: a distributed storage system for structured data," in *Operating Systems Design and Implementation*, 2006, pp. 205–218.

[14] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Symposium on Operating System Design and Implementation*, 2004.

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

[16] R. Vilaça, F. Cruz, and R. Oliveira, "On the expressiveness and trade-offs of large scale tuple stores," in *On the Move to Meaningful Internet Systems, OTM 2010*, ser. Lecture Notes in Computer Science, R. Meersman, T. Dillon, and P. Herrero, Eds. Springer Berlin / Heidelberg, 2010, vol. 6427, pp. 727–744. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-16949-6_5

[17] R. Vilaca and R. Oliveira, "Clouder: A flexible large scale decentralized object store: Architecture overview," 2009.

[18] R. Vilaca, R. Oliveira, and J. Pereira, in *International Conference on Distributed Applications and Interoperable Systems (DAIS11)*, Iceland, 2011.

[19] P. Eugster, R. Guerraoui, S. Handurukande, P. Kouznetsov, and A.-M. Kermarrec, "Lightweight probabilistic broadcast," *ACM Transactions on Computer Systems*, vol. 21, no. 4, pp. 341–374, 2003.

[20] J. Pereira, L. Rodrigues, R. Oliveira, and A.-M. Kermarrec, "Neem: Network-friendly epidemic multicast," in *Proceedings of the 22nd Symposium on Reliable Distributed Systems*. IEEE, 2003, pp. 15–24.

[21] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal Multicast," *ACM Transactions on Computer Systems.*, vol. 17, no. 2, pp. 41–88, 1999.

[22] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *Lecture Notes in Computer Science*, vol. 2218, 2001, pp. 329–350.

[23] J. Cardoso, C. Baquero, and P. Almeida, "Probabilistic estimation of network size and diameter," in *Dependable Computing, 2009. LADC '09. Fourth Latin-American Symposium on*, 2009, pp. 33 –40.

[24] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks: algorithms and evaluation," *Performance Evaluation In P2P Computing Systems*, vol. 63, no. 3, pp. 241–263, 2006.

[25] L. Massoulié, E. Le Merrer, A.-M. Kermarrec, and A. Ganesh, "Peer counting and sampling in overlay networks: random walk methods," in *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 2006, pp. 123–132.

[26] J. Sacha, J. Napper, C. Stratan, and G. Pierre, "Adam2: Reliable Distribution Estimation in Decentralised Environments," *2010 IEEE 30th International Conference on Distributed Computing Systems*, pp. 697–707, 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5541632

[27] M. Haridasan and R. van Renesse, "Gossip-based distribution estimation in peer-to-peer networks," p. 13, Feb. 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1855641.1855654

[28] P. Jesus, C. Baquero, and P. Almeida, *Fault-Tolerant Aggregation for Dynamic Networks*, 2010.

[29] S. Voulgaris, E. Rivi, A.-m. Kermarrec, and M. V. Steen, "Sub-2-Sub : Self-Organizing Content-Based Publish Subscribe for Dynamic Large Scale Collaborative Networks," *Event (London)*.

[30] D. Tran and C. Pham, "PUB-2-SUB: A Content-Based Publish/Subscribe Framework for Cooperative P2P Networks," *NETWORKING 2009*, vol. 5550, pp. 770–781, May 2009. [Online]. Available: http://www.springerlink.com/index/1705131P5411181W.pdf

[31] R. Chand and P. Felber, "Semantic peer-to-peer overlays for publish/subscribe networks," in *Proceedings of the International Conference on Parallel and Distributed Computing (Euro-Par'05)*, Aug. 2005.

[32] M. Jelasity, A. Montresor, and O. Babaoglu, "T-Man: Gossip-based fast overlay topology construction," *Computer Networks*, vol. 53, no. 13, pp. 2321–2339, Aug. 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1570533.1570626

[33] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication," in *DEBS '07: Proceedings of the 2007 inaugural international conference on Distributed event-based systems*. New York, NY, USA: ACM, 2007, pp. 14–25.

[34] M. Matos, A. Nunes, R. Oliveira, and J. Pereira, "Stan: Exploiting shared interests without disclosing them in gossip-based publish/subscribe," in *9th International Workshop on Peer-to-Peer Systems (IPTPS10)*, April 2010.

[35] D. Peng and F. Dabek, "Large-scale incremental processing using distributed transactions and notifications," in *USENIX Conference on Operating Systems Design and Implementation*, 2010, pp. 1–15.

[36] The Apache Software Foundation, "HBase," 2011. [Online]. Available: http://hbaseblog.com/2010/11/30/hbase-coprocessors/

[37] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems*, vol. 23, no. 3, pp. 219–252, 2005.