Ability: Trade Lock in to Meta Storage

[1] **T. Leelavathi**, [2] **U.Usha Rani**

[1]M.Tech Research Scholar, Department of CSE,
[2] HOD, Department of CSE
Priyadarshini Institute of Technology & Science, Chintalapudi, India

Abstract:-*Expense and versatility advantages of Cloud storage administrations are evident. Notwithstanding, selecting a solitary storage service provider limits accessibility and versatility to the chose supplier and may further cause a merchant lock basically. In this paper, we introduce Meta Storage Capacity, and unified Cloud storage framework that can incorporate differing Cloud storage suppliers. Meta storage Capacity is a very accessible and versatile circulated hash table that imitates information on top of assorted storage service. Meta Capacity reuses instruments from Amazon's Dynamo for cross-supplier replication and thus acquaints a novel methodology with oversee consistency-inactivity tradeoffs by amplifying the customary majority (N; R; W) configurations to an (np; R; W) conspire that incorporates distinctive providers as an extra measurement. With Metastorage, new means to control consistency-inertness tradeoffs are presented.*

Key Terms: Trade Lock, Metastorage, Coordinator Bootstrapping.

I.INTRODUCTION

Cloud Computing is an emerging technology used to deliver on demand services over the Internet. It is undoubtedly affecting the way business is conducted and is empowering a new Generation of products and services. Cloud Computing can be summarized into three keywords: elasticity, on-demand, and (autonomously) fully-managed. These three characteristics massively benefit organizations by reducing both CAPEX and OPEX while enabling them to channel their efforts to the strategic business sector. Over the last few years the Cloud Computing market has grown tremendously and frequent new service offerings are emerging steadily. In particular, there is a large number of Cloud storage services, each focusing on different capabilities and guarantees. To satisfy availability and scalability needs of most Cloud-based applications, NoSQL databases have be-come very popular, owning the highest share of Cloud storage offerings [1], [2]. Besides, in-memory databases or Cloud relational database clusters are common alternatives, providing high-performance and consistency guarantees respectively. The choices are many, but vendor lock-in is still an issue as Cloud storage offerings tie customers to one particular offering due to immense switching costs for data migration. The remainder of the paper is structured as follows: First, we introduce Meta Storage, a Cloud storage federation system, and describe its design and implementation details as well as the additional parameters we introduced to balance consistency latency tradeoffs. Afterwards, we present the results of a system evaluation regarding consistency, availability and latency. Finally, we discuss the system's weaknesses and strengths and end with a conclusion.

II. RELATED WORK

There is preliminary work on Cloud storage systems to overcome vendor lock-in and to improve availability of stored data. Bunch et al. [23] extended the App Scale platform with unified access to diverse Cloud storage services using the Google App Engine Storage API. App Scale, however, can only connect to one data store and applications deployed on the platform are restricted to this connection. Bromberg et al. [24], [25] leverage multiple Cloud storage systems to increase the performance of content delivery with a Meta Content Delivery Network (Meta CDN) and developed a prototype to evaluate performance gains of their approach. Meta CDN focuses on read performance needed for fast content delivery and, therefore, replicates data to many Cloud storage services. To improve reads Meta CDN routes each content request to the replica available with the lowest expected latency. Meta CDN, however, lacks support for adequate write performance and immediate replication and, thus, cannot be employed as a full-fledged storage system. Similarly, Bowers et al. [15] developed a High Availability and Integration Layer (HAIL) that stores data in encrypted files Cloud over multiple storage services and returns decrypted data upon read requests with low compute effort. HAIL improves data security by utilizing encryption and data distribution over multiple Cloud storages but disregards scalability and introduces a bottleneck as it excludes a component comparable to our Coordinator. With Redundant Array of Cloud Storage (RACS) Abu Libode et al. [26] propose a Cloud storage overlay system which acts as a proxy that uses erasure coding [32] to distribute files over multiple Cloud storages, simulating a Redundant Array of Independent Disks (RAID) system. However, every write operation terminates only when all Cloud storage services have completed the operation, leading to high latencies for data that is Cloud world-wide. Furthermore, as RACS is not based on full replication it requires huge numbers of storage offerings which might not even exist in the first place. Also, built on top of eventually consistent [28] storage services RACS might fail in retrieving any data at all while other systems should at least return an outdated version.

III.OVER VIEW OF METASTORAGE

MetaStorage is a highly scalable, highly available, Cloud hash table, layered on top of different Cloud storage providers. For this purpose, MetaStorage reuses mechanisms from Amazon's Dynamo [3], but elevates these for cross provider data replication to maximize scalability, availability, vendor independence and fault tolerance. MetaStorage replicates data across several providers despite using machines of a single provider only. By integrating diverse Cloud storage providers, MetaStorage extends traditional quorum systems that use (N; R;W) configurations to balance not only consistency-availability but also to balance consistency-latency tradeoffs. Now, we can still use (N; R; W)but also add the dimension of providers as an additional knob to tweak consistency-latency tradeoffs. We suggest novel (NP; R; W)configurations where $N_P \geq 1$ is the total number of replica N_P that is hosted with the set of nproviders.The providers host a set of replica, formally defined as $N_P=N_1 \cup ::: \cup Nn$ where each provider I $\in$\{1……n\} ghosts $|N_i|$ replica.

A. Meta Storage Architecture

The Meta Storage architecture (figure 1) is based on nodes which act as wrappers for Cloud storage services. A set of nodes is aggregated within a Distributor which includes all functionality to replicate and retrieve data as well as assert availability of replica. To avoid the Distributor becoming a bottleneck we attached a Coordinator component to each Distributor which is responsible for periodically exchanging state between Distributors. Meta Storage components internally communicate using an asynchronous messaging protocol which can be seen as a subset of the staged event-driven architecture (SEDA) [4]. The main advantage of SEDA is that it degrades gracefully under heavy load as the overhead for thread synchronization stays constant no matter how many requests have to be processed per second. This is the reason why it was also internally used within Dynamo and reused in our context. The Meta Storage architecture (figure 1) is based on nodes which act as wrappers for Cloud storage services. A set of nodes is aggregated within a Distributor which includes all functionality to replicate and retrieve data as well as assert availability of replica. To avoid the Distributor becoming a bottleneck we attached a Coordinator component to each Distributor which is responsible for periodically exchanging state between Distributors. Meta Storage components internally communicate using an asynchronous messaging protocol which can be seen as a subset of the staged event-driven architecture (SEDA) [4]. The main advantage of SEDA is that it degrades gracefully under heavy load as the overhead furthered synchronization stays constant no matter how many requests have to be processed per second. This is the reason why it was also internally used within Dynamo and reused in our context.

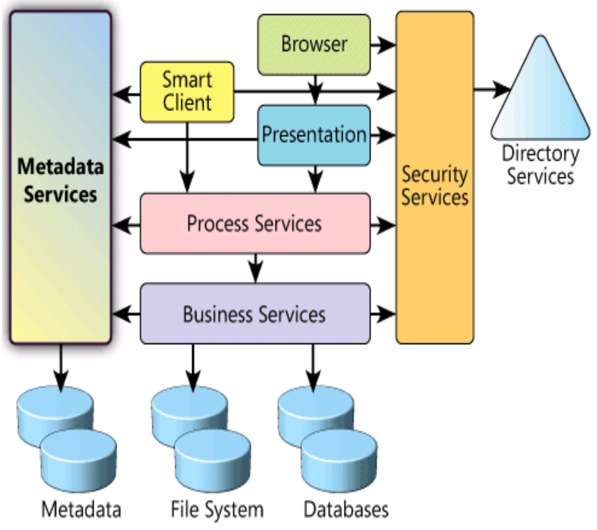

Fig.1. Meta Storage architecture

B.Meta Storage Nodes

Within the Meta Storage system the nodes are situated at the lowest layer. Their most important task is to offer a generic interface to the Distributor so that all technical details of the underlying infrastructure are hidden. Thus, they are basically wrappers for Cloud storage services like Amazon S3 (see also [5] and [6]). So far, we have built nodes for Amazon S3 (which can also connect to Walrus [7]), Google App Engine (plus the corresponding service running there, which is compatible to App Scale [8]), and for local hard disks. Further nodes are planned. In theory, there are no system limitations to extensibility.

Every node shares two message queues with its corresponding Distributor. Incoming messages are checked for their request type and then mapped to the respective methods which return a response message. The methods provided by the nodes are GET, PUT, LISTFILES and DELETE of which each node assumes that they can be invoked multiple times simultaneously, i.e. all synchronization issues on this level are pushed to the underlying infrastructure services.

C.Distributor

The Distributor is situated in the second layer from the bottom and it is the component within Meta Storage which is "doing the actual work". The Distributor alone is responsible for replication and retrieval of files. All components on a higher layer are usually granted fragmentation transparency. Requests to the Distributor are also sent asynchronously for which purpose every Distributor holds an input and output queue. All operations offered by the Distributor are idempotent, so, if an error occurs one may just resend the request.

The Distributor implementation bases its distribution mechanism on an approach presented by DeCandiaetal. [3] as well as Lakshman and Malik [9] which describes the concept of a preference list based on the hash of the key. Depending on the preferred Cloud storage services and their order within the preference list files are stored on the first N nodes and, thereby, Cloud to multiple providers. Since Meta Storage is a quorum-based system [10] already R successful reads (and W for writes respectively) are sufficient to return success. Whereas in Dynamo the preference list originally contained physical and later on logical nodes we adopted and changed the approach to fit into our scenario: N, R and W can still be configured but the preference list is identical for the entire key range, which makes sense because an entire Cloud storage service is less likely to fail than a single machine and is also expected to have a load balancing scheme of its own. We, thus, have no need for partitioning algorithms like consistent hashing [11], [12]. So, every Distributor instance contains a preference list which is an ordered list of Meta Storage nodes. Changes to the preference list and the (N_P; R; W) configuration are also possible at runtime. Whenever this is done the system halts and waits until all active requests have germinated. Upon completion the changes are applied and all processes get restarted. Apart from removing the need of partitioning algorithms the static preference list also gives us the second knob to balance consistency-latency tradeoffs awe already pointed out. There is one difference to [10], though: Quorum-based systems usually require a configuration where R+W > N to avoid reading stale data as well as W >N/2to avoids conflicts arising from concurrent writes. Since this also affects availability these requirements have been ignored in both Dynamo and MetaStorage.In the following we will present the design of the GET and PUT operations of the Distributor. See also table I for a brief overview of all supported operations.

PUT: Whenever the Distributor receives a PUT request it rebroadcasts it to the first N nodes of the preference list. Afterwards, the Distributor waits for responses. Whenever a response is of type error a new PUT request is created and sent to the next node of the preference list which has so far not

TABLE I OPERATIONS

As soon as W nodes have returned a success message the PUT operation terminates and responds to the requester. But while $W < N_P$ the system continues in the Background to bring up the number of replica from W to N. In any case, if less than W nodes respond with success and every node has already been contacted, an error message is returned. When the file has finally been stored on N nodes the system checks whether those N nodes are identical to the first N nodes of the preference list. If not so-called Hinted Handoffs [3] are created and kept locally in memory. A Hinted Handoff contains three pieces of information: The node of the first N nodes of the preference list which reported an error, the node which stored the file instead and the affected file key .The Distributor includes several sub processes which periodically try to resolve the existing Hinted Handoffs. Details are beyond the scope of this paper. There is one special case for which we have not been able to find a solution so far: If W nodes acknowledge storing the data but all other nodes in the preference list fail, a success message has already been returned because the algorithm could not know in advance that it would not be possible to bring the number of replica up to N. So far, there will not be more than W replica until the point where more nodes are available again And another GET or a PUT request is issued. To reduce the chance of such a situation occurring we propose sufficiently long preference lists combined with a few local file system nodes to cache the data in between.

2) GET: Whenever GET is invoked the operation retrieves the preference list and queries the list of Hinted Handoffs. Based on the request's key an updated temporary preference List is created which contains all N nodes which hold a copy of the requested file. Future versions might query the first R healthy nodes in the absence of Hinted Handoffs. This would allow a lazier synchronization with other Distributor instances. Next, messages containing GET requests for the respective key are sent to all nodes on the temporary preference list. Afterwards, the Distributor waits for the node's responses. Further Operations: Apart from GET and PUT Meta Storage also provides two operations to list all stored files (comparable to the Linux command less or the DOS command dir) as well as to delete specific files. We propose to choose one of the two versions based on the specific use case. For more information on all operations see table I. In section V- A we discuss the latency-consistency tradeoffs which can be addressed by choosing among the two delete operations DELETE and ASSERTEDDELETE. This small knob exists independent of the provider selection.

A. Coordinator

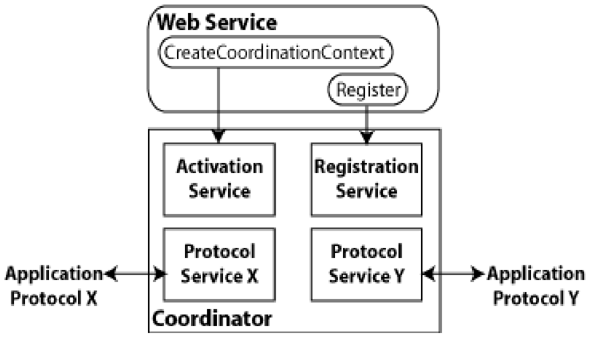

Fig.2.Overview of Coordinator Bootstrapping

When we combine a Distributor instance with some nodes we already have a running system which processes incoming messages, evaluates and executes their requests and returns responses. There is one issue, though: We are in a highly scalable environment and every underlying storage infrastructure is presumed to be scaling as well (Elson and Howell [13] reason why scalability is so much of importance). But if we use only one Distributor we will create a perfect bottleneck in our application landscape. To avoid this, we thought about adding independent Distributor instances but quickly discovered that some coordination between them is necessary. For example, every Distributor should have the same preference list and (N_P; R; W) configuration. Also, with every PUT request the set of Hinted Handoffs might change but other instances would not know about it. So, we finally added another layer on top of the Distributor: the Coordinator. Essentially, the task of our Coordinator is to manage the state of the underlying Distributors and to keep them all up-to-date in terms of configuration or membership changes. Figure 1 shows how Meta Storage becomes scalable by the use of Coordinators.

In order to avoid a fully centralized system but also, for ease of implementation, a completely decentralized system we propose a semi-decentralized solution: There is a master Coordinator which determines all other Coordinator's state. Now we had to cope with failing master Coordinators instead and solved this by giving every Coordinator a complete ordered list of all Coordinators within a system. Whenever the master cannot be reached for a certain period of time the remaining Coordinators each assumes the master to be offline and remove it from their list of Coordinators (Lindsay [14] provides arguments in favor of local action in case of failures). Thus, the former No. 2 becomes the new No. 1 and master. Since Coordinators know about all other Coordinators and Their specific order every one of them can decide – without central control – which the new master is as well as when it becomes a master.New Coordinators are always appended to the list of Coordinators so that the list is ordered by the total length of server uptime. This guarantees that every Coordinator which knows of more than three Coordinators (the master, some other Coordinator and itself) always knows No. 2. So, this implies that – when the master fails – every Coordinator which was not only known to the master before it failed also knows about No.2.

This leaves only two issues:

1) What happens if a Coordinator registers with the master but the master fails before it can respond?

2) What happens if a Coordinator registers with the master, the master responds but fails before it can forward the **Meta Storage Host:**

Surrounding the Coordinator there is an entire collection of utility classes or functions. One of the most useful ones is the Meta Storage Host. Basically, it is a local registry for local Meta Storage instances and, hence, allows running more than one Coordinator-Distributor pair within the same Java Virtual Machine. This could be useful to fully take advantage of machines with lots of CPU cores. Since all instances are identified by unique IDs a Meta Storage Host can forward incoming requests to the specific instance associated with the ID.

Apart from its function as a registry the Meta Storage Host is also responsible for information and functionality shared by all Coordinators running within the same Java VM. This includes hosting the Web Service interfaces as well as handling all incoming and outgoing requests for which it also provides parameter transformations, syntax checks and authentication. Furthermore, the host includes message handlers to map from synchronous SOAP requests to asynchronous internal messaging. Future versions might also allow asynchronous SOAP requests with callbacks.

F. **Security:**

Security measures in Meta Storage include a role-based user management which allows distinguishing between different rights as well as several security levels with the corresponding demands on the system and (as a future extension) the option to enable encryption before persisting data in the Cloud.

While some nodes already communicate via https every single Web Service call is still unencrypted. This is due to limitations of the used JAX-WS implementation which only supports http. Of course, this critically affects security so that we plan to include another JAX-WS server implementation in future versions. Another aspect is file encryption: Right now, many enterprises avoid (public) Cloud offerings as internal guidelines forbid storing internal data off-premises. To offer Meta Storage also in this context it could easily be achieved that every file passing Meta Storage is encrypted before writing it to the Cloud, i.e. before it leaves the responsibility of the customer. The latter approach is also taken in other systems which are "paranoid" in the sense that they consider their storage nodes to be an, at least potentially, hostile environment. Examples include Far site [19], HAIL [20], Ocean store [21] or Antiquity [17].

IV.CONCLUSION

In this paper, we presented the design and implementation of the Meta Storage system, a federated architecture that utilizes diverse Cloud storage providers. Meta Storage implements a replication scheme based on Amazon's Dynamo, but elevates concepts to a network of (autonomous and heterogeneous) storage providers. We have shown that Meta Storage increases overall availability compared to any individual provider. Furthermore, Meta Storage introduces provider configurations (in preference lists) as a new means beyond existing configurations of traditional quorum systems and thus provides additional control mechanisms to manage consistency-latency tradeoffs.

REFERENCES

[1] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the Cloud? An architectural map of the Cloud landscape," in Software Engineering Challenges of Cloud Computing, 2009. CLOUD'09. ICSE Workshop on. IEEE, 2009, pp. 23–31.

[2] C. Baun, M. Kunze, J. Nimis, and S. Tai,Cloud Computing: Web-basierte dynamische IT-Services, ser. Informatik im Fokus. Berlin:Springer, 2010.

[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," inProc. SOSP, 2007

[4] M. Welsh, D. Culler, and E. Brewer, "SEDA: architecture forwell-conditioned, scalable Internet services,"ACM SIGOPS OperatingSystems Review, vol. 35, no. 5, pp. 230–243, 2001.

[5] S. Garfinkel, "An Evaluation of Amazon's Grid Computing Services:EC2, S3, and SQS," inCenter for. Citeseer, 2007

[6] A. T. Velte, T. J. Velte, and R. Elsenpeter,Cloud Computing: A Practical Approach. Upper Saddle River, NJ: McGraw-Hill, 2010.

[7] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Yous-eff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid. IEEE, 2009, pp.124–13

[8] N. Chohan, C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman,and R. Wolski, "Appscale: Scalable and open appengine applicationdevelopment and deployment,"First International Conference on CloudComputing, 2009.

[9] A. Lakshman and P. Malik, "Cassandra: a decentralized structuredstorage system,"ACM SIGOPS Operating Systems Review, vol. 44, no. 2,pp. 35–40, 2010.

[10] R. Thomas, "A majority consensus approach to concurrency controlfor multiple copy databases," ACM Transactions on Database Systems(TODS), vol. 4, no. 2, pp. 180–209, 1979

[11] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, andD. Lewin, "Consistent hashing and random trees: Cloud cachingprotocols for relieving hot spots on the World Wide Web," in Proceedings of the twenty-ninth annual ACM symposium on Theory ofcomputing. ACM, 1997, pp. 654–663.

[12] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan,"Chord: A scalable peer-to-peer lookup service for internet applications,"in Proceedings of the 2001 conference on Applications, technologies,architectures, and protocols for computer communications. ACM, 2001,pp. 149–160.

[13] J. Elson and J. Howell, "Handling flash crowds from your garage,"in USENIX 2008 Annual Technical Conference on Annual TechnicalConference. USENIX Association, 2008, pp. 171–184.

[14] S. Bourne, "A conversation with Bruce Lindsay,"Queue, vol. 2, no. 8,pp. 22–33, 2004.

[15] T. Chandra, R. Griesemer, and J. Redstone, "Paxos made live: anengineering perspective," inProceedings of the twenty-sixth annual ACMsymposium on Principles of Cloud computing. ACM, 2007, pp.398–407

[16] L. Lamport, "The part-time parliament,"ACM Transactions on ComputerSystems (TOCS), vol. 16, no. 2, pp. 133–169, 1998

[17] H. Weatherspoon, P. Eaton, B. Chun, and J. Kubiatowicz, "Antiquity:exploiting a secure log for wide-area Cloud storage,"ACM SIGOPSOperating Systems Review, vol. 41, no. 3, pp. 371–384, 2007.

[18] M. Burrows, "The Chubby lock service for loosely-coupled Cloudsystems," in Proceedings of the 7th symposium on Operating systemsdesign and implementation. USENIX Association, 2006, pp. 335–350.

[19] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur,J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer,

"FARSITE:Federated, available, and reliable storage for an incompletely trustedenvironment,"ACM SIGOPS Operating Systems Review, vol. 36, no. SI, pp. 1–14, 2002

[20] K. Bowers, A. Juels, and A. Oprea, "HAIL: A high-availability andintegrity layer for cloud storage," in Proceedings of the 16th ACMconference on Computer and communications security. ACM, 2009,pp. 187–198.

[21] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels,R. Gummadi, S. Rhea, H. Weatherspoon, C. Wellset al., "Oceanstore: An architecture for global-scale persistent storage,"ACM SIGARCH Computer Architecture News, vol. 28, no. 5, pp. 190–201, 2000