The Hardest Problems in Data Management

vmware®

Table of Contents

The Hardest Problems in Data Management

Modern hardware trends and economics [1] combined with cloud/virtualization technology [2] are radically reshaping today's data management landscape, ushering in a new era where many machine, many core, memory-based computing topologies can be dynamically assembled from existing IT resources and pay-as-you-go clouds. Arguably one of the hardest problems — and consequently, most exciting opportunities — faced by today's solution designers is figuring out how to leverage this on-demand hardware to build an optimal data management platform that can:

• Exploit memory and machine parallelism for low-latency and high scalability grow and shrink elastically with business demand

• Treat failures as the norm, rather than the exception, providing always-on service

• Span time zones and geography uniting remote business processes and stakeholders

• Support pull, push, transactional, and analytic based workloads increase cost-effectiveness with service growth

What's So Hard

A data management platform that dynamically runs across many machines requires — as a foundation — a fast, scalable, fault-tolerant distributed system. It is well-known, however, that distributed systems have unavoidable trade-offs and notoriously complex implementation challenges [3,4]. Introduced at PODC 2000 by Eric Brewer [5] and formally proven by Seth Gilbert and Nancy Lynch in 2002 [6], the CAP Theorem, in particular, stipulates that it is impossible for a distributed system to be simultaneously: Consistent, Available, and Partition-Tolerant. At any given time, only two of these three desirable properties can be achieved. Hence when building distributed systems, design trade-offs must be made.

# Eventually Consistent Solutions Are Not Always Viable

Driven by awareness of CAP limitations, a popular design approach for scaling Web-Oriented applications is to anticipate that large-scale systems will inevitably encounter network partitions and to therefore relax consistency in order to allow services to remain highly available even as partitions occur [7]. Rather than go down and interrupt user service upon network outages, many Web user applications--e.g., customer shopping carts, e-mail search, social network queries—can tolerate stale data and adequately merge conflicts, possibly guided by help from end users once partitions are fixed. Several so-called eventually consistent platforms designed for partition-tolerance and availability—largely inspired by Amazon, Google, and Microsoft—are available as products, cloud-based services, or even derivative, open source community projects.

In addition to these prior art approaches by Web2.0 vendors, GemStone's perspective on distributed system design is further illuminated by 25 years of experience with customers in the world of high finance. Here, in stark contrast to Web user-oriented applications, financial applications are highly automated and consistency of data is paramount. Eventually consistent solutions are not an option. Business rules do not permit relaxed consistency, so in variants like account trading balances must be enforced by strong transactional consistency semantics. In application terms, the cost of an apology [8] makes rollback too expensive, so many financial applications must limit workflow exposure traditionally by relying on OLTP systems with ACID guarantees.

But just like Web 2.0 companies, financial applications also need highly available data. So in practice, financial institutions must "have their cake and eat it too".

So given CAP limitations, "How is it possible to prioritize consistency and availability yet also manage service interruptions caused by network outages?"

The intersection between strong consistency and availability (with some assurance for partition-tolerance) identifies a challenging design gap in CAP-aware solution space. Eventually consistent approaches area non-starter. Distributed databases using 2 phase commit provide transactional guarantees but only so long as all nodes see each other. Quorum based approaches provide consistency but block service availability during network partitions.
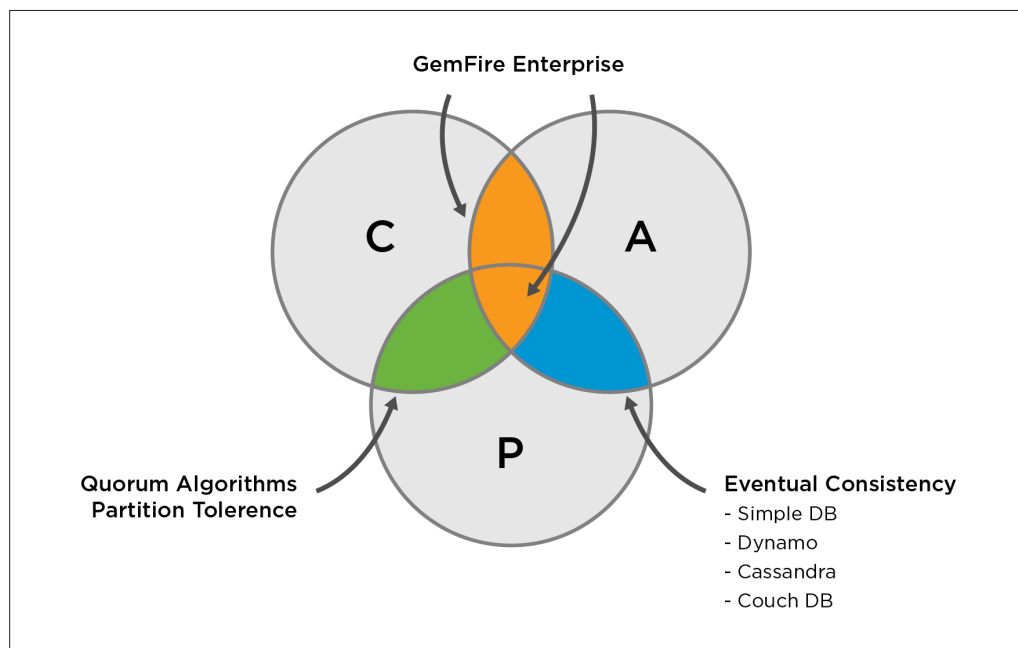
Collaboratively designed with customers over the last five years, GemStone has developed practical ways to mine this gap, delivering an off-the-shelf, fast, scalable, and reliable data management solution we call the enterprise data fabric (EDF).

OUR SOLUTION: THE EDF

Any choice of distributed system design—shared memory, shared-disk, or shared-nothing architecture—has inescapable CAP trade-offs with downstream implications on data correctness and concurrency. High-level design choices also impact throughput and latency as well as programmability models, potentially imposing constraints on schema flexibility and transactional semantics. Guided by distributed systems theory [9], industry trends in parallel/distributed databases [10], and customer experience, the EDF solution adopts a shared-nothing scalability architecture where data is partitioned onto nodes connected into a seamless, expandable, resilient fabric capable of spanning process, machine, and geographical boundaries. By simply connecting more machine nodes, the EDF scales data storage horizontally. Within a data partition (not to be confused with network partitions), data entries are key/value pairs with thread-based, read-your-writes [7] consistency. The isolation of data into partitions creates a service-oriented design pattern where related

partitions can be grouped into abstractions called service entities [11]. A service entity is deployed on a single machine at a time where it owns and manages a discrete collection of data — a holistic chunk of the overall business schema — hence, multiple data entries co-located within a service entity can be queried and updated transactionally, independent of data within another service entity.

Mining the Gap in CAP

From a CAP perspective, service entities enable the EDF to exploit a well-known approach to fault tolerance based on partial-failure modes, fault isolation, and graceful degradation of service [12]. Service entities allow application designers to demarcate units of survivability in the face of network faults. So rather than striving for complete partition-tolerance (an impossible result) when consistency and availability must be prioritized, the EDF implements a relaxed, weakened form of partition-tolerance that isolates the effects of network failures enabling the maximum number of services to remain fully consistent and available when network splits occur.

By configuring membership roles (reflecting service entity responsibilities) for each member of the distributed system, customers can orthogonally (in a light-weight, non-invasive, declarative manner) encode business semantics that give the EDF the ability to decide under what circumstances a member node can safely continue operating after a disruption caused by network failure. Membership roles perform application decomposition by encapsulating the relationship of one system member to another, expressing causal data dependencies, message stability rules (defining what attendant members must be reachable), loss actions (what to do if required roles are absent), and resumption actions (what to do when required roles rejoin). For example, membership roles can identify independent subnetworks in a complex workflow application. So long as interdependent producer and consumer activities are present after a split, reads and writes to their data partitions can safely continue. Rather than arbitrarily disabling an entire losing side during network splits, the EDF consults fine-grained knowledge of service-based role dependencies and keeps as many services consistently available as possible.

Have It All (Just Not at Once)

CAP requirements and data mutability requirements can evolve as data flows across space and time through business processes.

So all data is not equal[13]—as data is processed by separate activities in a business workflow, consistency, availability, and partition-tolerance requirements change. For example, in an e-commerce site, acquisition of shopping cart items is prioritized as a high-write availability scenario, i.e., an e-retailer wants shopping cart data to be highly available even at the cost of weakened consistency, lest service interruption stalls impulse purchases, or worse yet, ultimately forces customers to visit competitor store fronts. Once items are placed in the shopping cart and the order is clicked, automated order fulfillment workflows reprioritize for data consistency (at the cost of weakened availability) since no live customer is waiting for the next Web screen to appear; if an activity blocks, another segment of the automated workflow can be alternatively launched, or the activity can be retried. In addition to changing CAP requirements, data mutability requirements also change— e.g., at the front end of a workflow, data may be write-intensive; whereas afterward, during bulk processing, the same captured data may be accessed in read-only or read-mostly fashion.

Our EDF solution exploits this changing nature of data by flexibly configuring consistency, availability, and partition-tolerance trade-offs, depending on where and when data is processed in application workflows:

Thus, business can achieve all three CAP properties—but at different application locations and times.
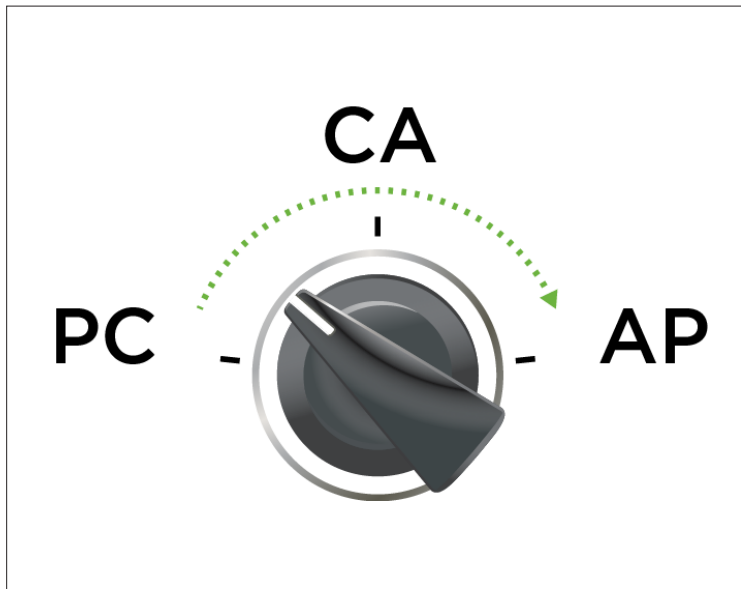
Each logical unit of EDF data sharing, called a region, maybe individually configured for synchronous or asynchronous state machine replication, persistence, and stipulations for N (number of replicas), W (number of writers for message stability), R (number of replicas for servicing a read request).

Amortizing CAP For High Performance

In addition to tuning CAP properties, this system configurability lets designers selectively amortize the cost of fault-tolerance throughout an end-to-end system architecture resulting in optimal throughput and latency.

For example, Wall Street trading firms with extreme low-latency requirements for data capture can configure order matching systems to run completely in-memory (with synchronous, yet very fast, redundancy to another in-memory node) while providing disaster recovery with persistent (on-disk), asynchronous data replication and eventual consistency to a metropolitan area network across the river in New Jersey. To maximize performance for competitiveness, risks of failure are spread out and fine-tuned across the business workflow according to failure probability: the common failure of single machine nodes is offset by intra-datacenter, strongly consistent HA memory backups; while the more remote possibility of a complete data center outage event is off set by a weakly consistent, multi-homed, disk-resident disaster recovery backup.

Exploiting Parallelism

A design goal of the EDF is to optimize performance by exploiting parallelism and concurrency wherever and whenever possible. For example, the EDF exploits many-core thread-level parallelism by managing entries in highly concurrent data structures and utilizing service pools for computation and network I/O. The EDF employs partitioned and pipelined parallelism to perform distributed queries, aggregation, and internal distribution operations on behalf of user applications. With data partitioned among multiple nodes, a query can be replicated to many independent processors each returning on a small part of the overall query. Similarly, business data schemas and workloads frequently consist of multi-step operations on uniform (e.g., sequential time series) data streams. These operations can be composed into parallel dataflow graphs where the output of one operator is streamed into the input of another; hence multiple operators can work continuously in task pipelines.
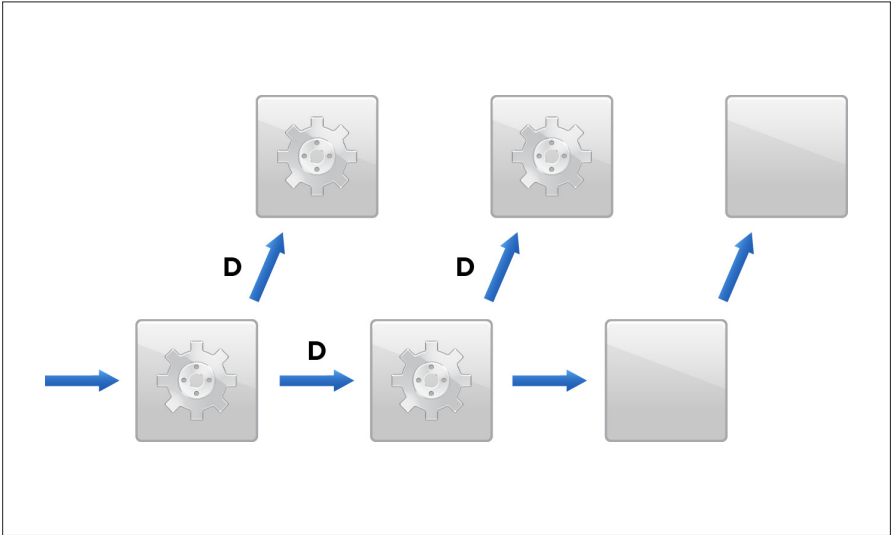
External applications can use the Function Service API to create Map/Reduce programs that run in parallel over data in the EDF. Rather than data flowing from many nodes into a single client application, control (Runnable functions) logic flows from the application to many machines in the EDF. This dramatically reduces process execution time as well as network bandwidth.
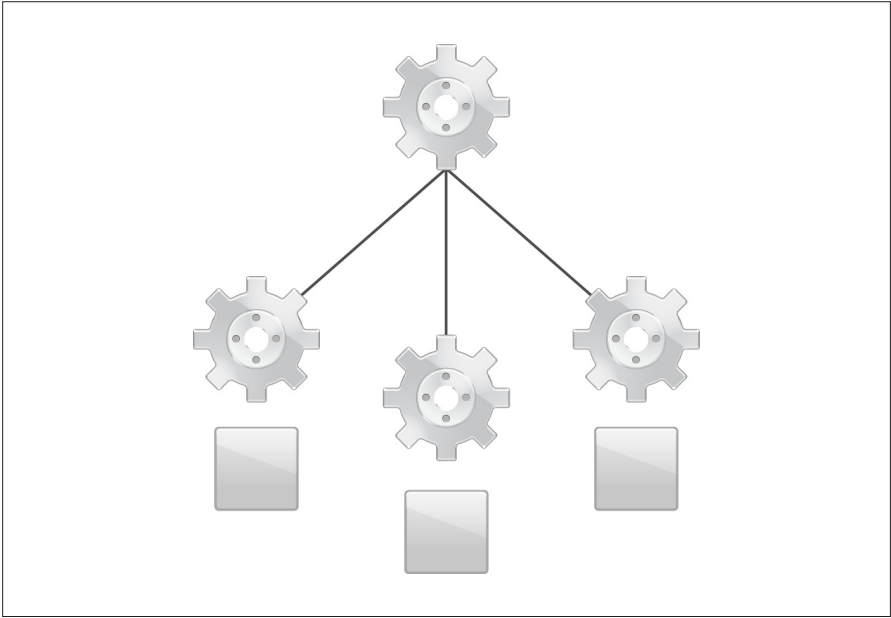
To further assist parallel function execution, data partitions can be tuned dynamically by all applications using the Partition Resolver API. By default, the EDF uses a hashing policy where a data entry key is hashed to compute a random bucket mapped to a member node. The physical location of the key-value pair is virtualized from the application. Custom partitioning, on the other hand, enables applications to co-locate related data entries together to form service entities. For example, a financial risk analytics application can co-locate all trades, risk sensitivities, and reference data associated with a single instrument in the same region. Using the

Function Service, control flow can then be directly routed to specific nodes holding particular data sets; hence queries can be localized and then aggregated in parallel increasing the speed of execution when compared to a distributed query.

Server machines can also be organized into server groups — aligned on service entity functionality — to provide concurrent access to shared data on behalf of client application nodes.

Active Caching: Automatic, Reliable Push Notifications for Event-Driven Architectures
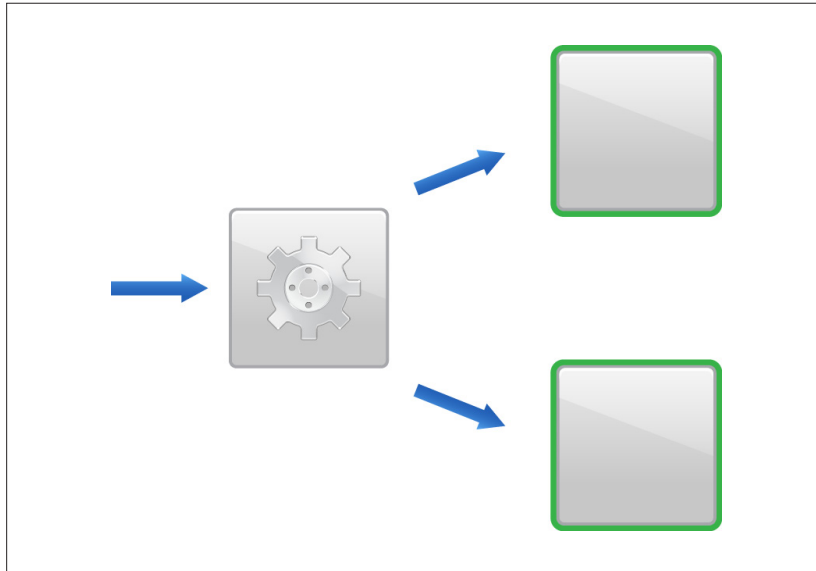
EDF client applications can then create active caches to eliminate latency. If application requests do not find data in the client cache, the data is automatically fetched from the EDF by the server. As modifications to shared regions occur, these updates are automatically pushed to clients where applications receive real-time event notifications. Likewise, modifications initiated by a client application are sent to the server and then pushed to other applications listening for region updates.

Caching eliminates the need for polling since updates are pushed to client applications as real-time events

In addition to subscribing to all events on a data region, clients can subscribe to events using key-based regular expressions or continuous queries based on query language predicates. Clients can create semantic views or narrow slices of the entire EDF that act as durable, stateful pub/sub messaging topics specific to application use-cases. Collectively, these views drive an enterprise-wide, real-time event-driven architecture.

Spanning The Globe

To accommodate wider-scale topologies beyond client/server, the EDF spans geographical boundaries using WAN gateways that allow businesses to orchestrate multi-site workflows. By connecting multiple distributed systems together, system architects can create 24x7 global workflows wherein each distinct geography/city acts as a service entity that owns and distributes its regional chunk of business data. Sharing of data across cities — e.g., to create a global book for coordinating trades between exchanges in New York, London, and Tokyo — can be done via asynchronous replication via persistent WAN queues. Here, consistency of data is weakened and eventual, but with the positive trade off of high application availability (the entire global service won't be disrupted if one city goes dark) and low-latency business processing at individual locations.

Memory-Centric Performance

With plunging cost of fast memory chips (1TB/$15,000), 64-bit computing, and multi-core servers, the EDF can give system designers an opportunity to rethink the traditional memory hierarchy for data management applications. The EDF obviates the need for high-latency (tens of milliseconds) disk-based systems by pooling memory from many machines into an ocean of RAM, creating a fast (microsecond latency), redundant virtualized memory layer capable of caching and distributing all operational data within an enterprise.

The EDF core is built in Java where garbage collection of 64-bit heaps can cause stop-the-world performance interruptions. Object graphs are managed in serialized form to decrease strain on the garbage collector. To reduce pauses further, there source manager actively monitors Java virtual machine heap growth and proactively evicts cached data on an LRU basis to avoid GC interruptions.

As flash memory replaces disk—an inevitability predicted by the new five minute rule [14]—EDF overflow and disk persistence can be augmented with flash memory for even faster overflow and lower-latency durability.

Elastic Scalability

From a distributed systems viewpoint, a key technical challenge for elastic growth is to figure out how to initialize new nodes into a running distributed system with consistent data. The EDF implements a patent-pending method for bootstrapping joining nodes based on the current data that ensures "in-flight" changes appear consistently in the new member nodes. When a node joins the EDF, it is initialized with a distributed snapshot [15] taken in a non-blocking manner from current state gleaned from the active nodes. While this initial snapshot is being delivered and applied at the new node, concurrent updates are captured and then merged once the new node is initialized.

Tools, Debugging, Testing, Support

Even with theorem proven algorithms and extensive hardware and development resources, implementing distributed systems is challenging. Service outages from cloud computing giants like Amazon, Google, and Microsoft attest to the enormous difficulty of getting distributed systems right.

Subtle race conditions occur within innumerable process interleavings spread across many threads in many processes in many machines. Recreating problem — at ic scenarios, messaging patterns, and timing sequences is difficult. Testing an asynchronous distributed system with a fail stop model is not even enough since Byzantine faults occur in the real world (e.g., TCP check sums fail silently).

Our EDF solution is continuously tested for both correctness and performance using state-of-the-art quality assurance and performance tools including static and dynamic model-checking, profiling, failure-injection, and code analysis. We use a highly configurable multi-threaded, multi-vm, multi-node distributed testing framework that provisions tests across a highly scalable hardware test bed. We run a continuous project to improve our modeling of EDF performance based on mathematical, analytical, and simulation techniques.

Despite rigorous design, testing, and simulation,we know that bugs will always exist. Therefore, a vital design consideration of the EDF distributed system Includes mechanisms for comprehensive management, debugging, and support, e..g, meticulous system logging, continuous statistics monitoring, system health checks, management consoles, scriptable admin tools, management APIs with configurable threshold conditions, performance visualization tools.

Our EDF solution is built and supported by a team with a 25+ year history of world-class 24x7x365 support that includes just-in-time, on-site operational consultations along with company-wide escalation path ways to insure customer success and business continuity.

SOLUTION	CONSISTENCY	TURNABLE CAP	MULTI-ENTRY TRANSACTIONS	LATENCY BOTTLENECK
EDF	Tunable: weak, read-your-writes, and ACID on Service-entities	Yes	Tunable: Linnearizability to Seriallizability	Memory (u Seconds)
Dynamo, Cassandra, Voldemort	Weak/Eventual	Tunable N/R/W	No: Only single key/ value updates	Disk (Dynamo=2ms, Cassandra=.12ms read, 15ms write, Voldemort=10ms)
SimpleDB	Weak/Eventual	Tunable N/R/W	No: Only single key/ value updates	Web (Seconds)
GoogleApp Engine/ Megastore	Strong on local entitiesgroups; Weak/Eventual for distributed updates	No	Yes	Disk (30ms)
CouchDB	Weak/Eventual MVCC	No	Document Versioning	Disk (ms to sec)
Azure Tables	Basically available Soft State Eventual Consistencey (BASE)	No	Entity Group Transactions	Web (Seconds)
memcached	Weak/Eventual Expiration	No	No: Only single key/ value updates	Memory (u Seconds)

SOLUTION	ACTIVE CACHING	PUB/SUB CQS/ TRIGGERS	CUSTOMAPP PARTITIONING	FUNCTION SERVICE, MAP/REDUCE
EDF	Yes	Yes	Yes	Yes
Dynamo, Cassandra, Voldemort	No	No	No	No
SimpleDB	No	No	No	No

SOLUTION	ACTIVE CACHING	PUB/SUB CQS/ TRIGGERS	CUSTOMAPP PARTITIONING	FUNCTION SERVICE, MAP/REDUCE
GoogleApp Engine/ Megastore	No	No	No	No
CouchDB	No	Yes	No	Yes
Azure Tables	No	No	No	No
memcached	No	No	No	No

Conclusion

To build a data management platform that can scale to exploit on-demand virtualization environments requires a distributed system at its core. The CAP Theorem, however, tells us that building large-scale distributed system requires unavoidable trade offs. There is a current, design gap in CAP-aware solution space unsolved by the current wave of eventually consistent platforms. To bridge this gap, our solution, called the Enterprise Data Fabric (EDF), prioritizes strong consistency and high availability by introducing a weakened form of partition tolerance that lets businesses shrink application exposure to failure by demarcating functional boundaries according to service entity role dependencies. When network faults inevitably occur, failures can be contained in that service does not disappear wholesale, but degrades gracefully in a partial manner. Our solution also lets designers apply varying combinations of consistency, availability, and partition-tolerance at different times and locations in a business workflow. Hence, by tuning CAP semantics, a data solutions architect can amortize partition out age risks across the entire work flow, thus maximizing consistency with availability while minimizing disruption windows caused by network partitions. This configurability ultimately enables architects to build an optimal, high performance data management solution capable of scaling with business growth and deployment topologies.

From a functional view point, the EDF is a memory-centric distributed system for business-wide caching, distribution, and analysis of operational data. It lets businesses run faster, reliably, and more intelligently by co-locating entire enterprise working data sets in memory, rather than constantly fetching this data from high-latency disk.

The EDF melds the functionality of scalable data base management, messaging, and stream processing into holistic middleware for sharing, replicating, and querying operational data across process, machine, and LAN/ WAN boundaries. To applications, the EDF appears as a single, tightly interwoven, overlay mesh that can expand and stretch elastically to match the size and shape of a growing enterprise. And unlike single-purpose, poll-driven, key-value stores, the EDF is alive platform — real-time Web infrastructure — that pushes real-time events to applications based on continuous mining of data streams — this gives enterprise customers instant insight to perishable opportunities.

Designed, built, and tested for over the last five years on behalf of Wall Street customers, the EDF offers any mission-critical business with large volume, low latency, high consistency requirements, the chance to be more scalable, speedier, and smarter.

References

[1] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, John Shalf, S. Williams, K. Yelick, "The Landscape of Parallel Computing Research: A View from Berkeley", EECS Department University of California, Berkeley, Technical Report No. UCB/EECS-2006-183, December 18, 2006.

[2] D. Patterson, etal., Above the Clouds: ABerkeley View of Cloud Computing, http://d1smfj0g31qzek. cloudfront.net/abovetheclouds.pdf, 2009.

[3] S.Kendall, J. Waldo, A. Wollrath, G. Wyant, "A Note on Distributed Computing", http://research.sun.com/ technical-reports/1994/smli_tr-94-29.pdf, 1994.

[4] T. Chandra, R. Griesemer, J. Redstone, "Paxos Made Live – An Engineering Perspective", POD C '07:26th ACM Symposium on Principles of Distributed Computing, 2007.

[5] E.Brewer, POD C Keynote, http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC- keynote.pdf, 2000.

[6] S. Gilbert, N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent Available Partition-Tolerant Web Services", ACM SIG ACT News, 2002.

[7] W. Vogels, "Eventually Consistent", ACM Queue, 2008.

[8] P. Helland, "Memories, Guesses, and Apologies", http://blogs.msdn.com/pathelland/archive/2007/05/15/ memories-guesses-and-apologies.aspx, 2007.

[9] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", ACM, 1978.

[10] S. Madden, "Data base parallelism choices greatly impact scalability", The Database Column: A multi-author blog on database technology and innovation, http://www.databasecolumn.com/2007/10/database-parallelism-choices.html, October 30, 2007.

[11] P.Helland, "Life beyond Distributed Transactions: an Apostate's Opinion", CIDR 2007, http://www-db.cs.wisc. edu/cidr/cidr2007/papers/cidr07p15.pdf, 2007 Fox, "Harvest, Yield, and ScalableTolerant Systems", HOTOS Proceedings of the The Seventh Workshop on Hot Topics in Operating Systems 1999.

[13] W .Vogels, "Availability & Consistency", http://www.infoq.com/presentations/availability-consistency, 2007.

[14] G. Graefe, "The Five-Minute Rule 20 Years Later", ACM Queue, 2009.

[15] M. Chandy, L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems", research.microsoft.com/users/lamPort/pubs/chandy.pdf, ACM, 1985.

vmware®