In Search of a General Cost Model for ACID Compliant Distributed Databases

What is the cost of switching from PostgreSQL to CockroachDB?

Matthew McGraw, Athabasca University
COMP 418 – Distributed Databases and Database Tuning

Abstract

This research aims to provide an initial quantification of the performance of a distributed ACID compliant SQL database compared to a traditional monolithic ACID compliant RDBMS. Specifically it compares a write-only, concurrent, OLTP workload across two database vendors – PostgreSQL and CockroachDB. Since CockroachDB offers a variety of deployment options, this research develops a multivariate model to assess the performance slowdown that occurs under several deployment options – notably a multi-node cluster within a single data center, a multi-node cluster across a geographically dense networked area, and a multi-node cluster spread across an intercontinental network. The results conclude that distributed databases take longer to perform write heavy workloads than monolithic RDBMS, and provide a generalized multivariate model that allows software development teams to estimate what slowdown rates to expect under different deployment architectures.

Introduction and Motivation

IBM's System R, widely considered the first SQL database, came to market in 1977[i] – just two years after the conclusion of the Vietnam War. In the 42 years that have followed, relational database management systems (RDBMS) have been the backbone of application development. Recently RDBMS have ushered in a new era of web applications, storing data of previously unthinkable volume. Without mentioning the proprietary titans like Oracle, Microsoft SQL Server, and IBM DB2, the mere open-source databases MySQL and PostgreSQL have scaled to data volumes that are nothing short of staggering – Facebook reaches 60M queries per second and 4M updates[ii], and all the way back in 2008 Yahoo was able to achieve 24B web events using a PostgreSQL[iii] database. However achieving this scale requires utmost engineering resources and focus, something not afforded to every web startup. Furthermore, there is a hard limit on how large a computer can be sourced, while there is a much software limit imposed on horizontally scaling – adding many smaller, commodity servers. Finally, there are good reasons to scale horizontally – redundancy and locality. Since traditional RDBMS systems are fundamentally designed to be single-node servers, they predominantly achieve redundancy through read replicas – duplicate copies of a database that follow in a step wise manner to a primary server. The problem of locality is rooted in the fact that as web applications grow in geographically diverse ways, a system is best served to allow users to connect to databases close to them, rather than sending network messages across a continent or an ocean. For these reasons, horizontal scaling, redundancy, and geographic distribution, we have seen the rise in popularity of NoSQL[1] databases.

While NoSQL databases offer many attractive features – infinite horizontal scaling, global distribution, and low latency reads and write - there is a catch to using NoSQL databases. Since NoSQL databases do not offer the same data guarantees that an ACID compliant SQL database would, often to the surprise of its users[iv]. This requires material changes to the application, often requiring changes to compensate for the lack of ACID compliance provided by the NoSQL database server. In fact, even the modern face of NoSQL usage, Google, released a prominent research paper hinting at the burden this places on application development[v]:

1 NoSQL Databases include Google BigTable, Amazon DynamoDB, MongoDB, Apache Cassandra – Hagerty, February 2018, CrateDB Blog, The Rise of the Distributed SQL Database https://crate.io/a/rise-distributed-sql-database/

> *Spanner exposes the following set of data features to applications: a data model based on schematized semi-relational tables, a query language, and general-purpose transactions. The move towards supporting these features was driven by many factors. The need to support schematized semi-relational tables and synchronous replication is supported by the popularity of Megastore [5]. At least 300 applications within Google use Megastore (despite its relatively low performance) because its data model is simpler to manage than Bigtable's, and because of its support for synchronous replication across datacenters. (Bigtable only supports eventually-consistent replication across data-centers.) Examples of well-known Google applications that use Megastore are Gmail, Picasa, Calendar, Android Market, and AppEngine. The need to support a SQL-like query language in Spanner was also clear, given the popularity of Dremel [28] as an interactive data-analysis tool. Finally, the lack of cross-row transactions in Bigtable led to frequent complaints; Percolator [32] was in part built to address this failing. Some authors have claimed that general two-phase commit is too expensive to support, because of the performance or availability problems that it brings [9, 10, 19].* ***We believe it is better to have application programmers deal with performance problems due to overuse of transactions as bottlenecks arise, rather than always coding around the lack of transactions. Running two-phase commit over Paxos mitigates the availability problems.***

Transactions, and therefore ACID compliance, greatly simply the complexity faced by application developers. Why then would we forgo them? Why wouldn't every distributed database implement ACID compliant transactions? Implementing distributed transactions comes with implementations that have increased latency and reduced throughput, due to the guarantees they provide[vi]. Simply put – it is easier to not implement ACID compliant transactions, enabling faster performance, which places the onus of ACID guarantees on the application programmers rather than the database provider.

CockroachDB takes a fundamentally different stance in the marketplace than it's NoSQL predecessors, and their marketing cuts straight to the core - "CockroachDB is ACID compliant, and scales globally without the need for a massive architectural overhaul. Simply add a new node to the cluster and CockroachDB takes care of balancing data across instances and geographies"[vii]. CockroachDB implements distributed transactions[viii] using Raft, a distributed consensus protocol and a variant of Paxos, to achieve a majority vote of agreement when coordinating a distributed write. For each write one node in the cluster is considered the primary node to determine if a write is legal (Leaseholder). If so, the leaseholder and raft leader will communicate to all other nodes with at least one networked round trip message. This message allows the nodes to vote on whether the write is conflict free and can be committed. If a majority of nodes vote to commit the transaction, the leaseholder will update its write-ahead log, committing the transaction and respond to the client. The transaction is considered durable at this point.

The goal of this research is to provide an initial step in attempting to provide a general model that allows software development teams to model their systems with and without a distributed database. For development teams that begin with a NoSQL system, ACID compliant or not, they have (hopefully) designed their applications to accommodate the features of their database. In a manner of speaking, they have no need for a general model of moving from a monolithic database server to a distributed one. For teams that have many years of experience building a system using a traditional monolithic RDBMS however, they cannot so easily make the switch from say, PostgreSQL to MongoDB, without rewriting a considerable amount of their application. This research aims to help development teams consider what is the cost of using an ACID compliant distributed database that does not force any

application changes. Using this principle, PostgreSQL and CockroachDB were chosen for several prominent reasons. First and foremost CockroachDB is compliant with the PostgreSQL wire protocol. This means that PostgreSQL drivers support CockroachDB out of the box, and offer the ability for an apples to apples performance test. Secondly, CockroachDB is fully open source, which increases the amount of knowledge available regarding their system architecture. Finally, CockroachDB has undergone a satisfactory Jepsen Analysis[ix], which provides independent validation and analysis to its product claims.

Related Work

Unfortunately there is scant research seeking to provide a standardized benchmark for distributed databases compared to monolithic databases. Perhaps this line of research falls more into the applied realm than into the academic, however there are small set of publications that have attempted similar lines of research. In 2000, Nicola and Jarke[x] provided a thorough survey of known techniques to analyze distributed databases, and also conducted several comparisons themselves. However, their work focused on monolithic database servers using various replication techniques, as these were common preceding the advent of NoSQL databases. Almost two decades later, Grim[xi] authored a paper at the University of Amsterdam analyzing CockroachDBs ability to remain consistent and durable under fault injection. This work sought to expand on Jepsen analysis, and found that under certain circumstances CockroachDB was not able to provide consistency and durability, two important factors of ACID compliance. A year following Grim, in 2017 Kaur[xii] provides a cross-sectional performance analysis of several NewSQL databases. This term is introduced as an evolution to NoSQL, distributed database systems which are mostly ACID compliant and mostly allow regular SQL commands (compared to an alternate or proprietary querying language). Kaur runs performance tests comparing CockroachDB, MemSQL, NuoDB, and VoltDB. However, the performance tests are limited in two important ways – first they are executed on single server machines (ie. the tests are not distributed in nature) and second the workloads are not available for inspection (or at least they were not detailed in the publication or cited for reference).

Performance Analysis

This research analysis uses a bespoke write heavy workload, designed to mimic a standard OLTP relation, and is made up of two major components – a software application and a database deployment.

The software application uses three relations from the Sean Lahman Baseball Database (http://www.seanlahman.com/baseball-archive/statistics/) in CSV format (“batting.csv”, “fielding.csv”, “pitching.csv”), the records are serialized into objects in the Go programming language and stored in a polymorphic container. Once the container is loaded with all the records, the records are processed using a variable number of concurrent workers to insert records into the database. The data sets are designed to prevent record contention, that is the writes do not conflict with one another, and there are no indexes to update. As noted earlier, given that CockroachDB uses the PostgreSQL wire protocol, a single driver is able to process records for both PostgreSQL and CockroachDB databases, varying only in the connection string provided at start up.

All results were generated using Digital Ocean as the infrastructure provider. Digital Ocean refers to their Virtual Private Servers (VPS) as Droplets and their data centers as Regions. The performance testing only used Standard Droplets with 2 vCPUs, 4GB RAM, 4TB of data transfer, and 80GB SSD hard drives, which cost $0.03 per hour. The operating system was Ubuntu 18.04 LTS. The databases were installed using the vendor installation instructions for Ubuntu, and the CockroachDB installation and deployment scripts are shown in the appendix. The underlying database deployments all vary according to the following schedule of parameters:

i. PostgreSQL database server, running on the same VPS as the software application. (single region Toronto)
ii. PostgreSQL database server, running on a separate VPS within the same data center as the software application. (single region Toronto)
iii. CockroachDB database server, running a single node cluster on the same VPS as the software application. (single region Toronto)
iv. CockroachDB database server, running as a three node cluster, each node on a separate VPS within the same region. (single region Toronto)
v. CockroachDB database server, running as a three node cluster, each node on a separate VPS within a different region. (multiple region New York City 1, New York City 3, Toronto)
vi. CockroachDB database server, running as a three node cluster, each node on a separate VPS within a different region, and each region within a different continent. (multiple region New York City 3, London, Singapore)

For each parameter schedule, three workloads were executed with varying concurrent workers, and each execution was repeated five times. Therefore a total of 90 workloads were executed. Prior to each workload a database migration was run to completely undo the effects of the previous runs, and it was verified that all records were executed as completed (i.e. no failed writes, etc.). The first two workloads were used to establish a control group, and given the low variability in performance (that is, the networked database server performed equal to the single server) the second group was chosen as the baseline comparison.

The appendix contains the source code for the Go binary, the various Bash scripts that invoked the executions, the resulting performance from each execution, and the configuration parameters for CockroachDB. The CockroachDB configurations increased cache and SQL memory sizes to 25% of system memory, and no adjustments were made for PostgreSQL.

Results

This research proposes a simple regression model to predict the percentage reduction in system throughput when moving to an ACID compliant distributed database. Plainly, we hypothesize that the percentage change in throughput can be explained by the number of concurrent workers, plus a series of dummy variables that specify whether we are using a multi node deployment, a multi-data center deployment, and finally an intercontinental deployment. Further notes are discussed in the Appendix under “Regression Logs”. The regression results shown below specify a log-linear regression, where the dependent variable is the natural log of the percentage increase of overall system runtime.

The results demonstrate a statistically significant performance costs to switching to an ACID compliant distributed database, and that these costs can be modeled using the log-linear specification to estimate the expected percentage increases in write throughout. Typically log-linear specifications are used to approximate percentage growth in tighter ranges, however in this case where the relative system runtime performances are at least twice as slow, the transformation supports smoothing the dependent variable to be appear linear. This still eases the interpretation and comparability of the regression results.

In short, a development team should expect at least a 60% reduction in throughput (an increase of 150% system runtime) moving to an ACID compliant distributed database. Pursuing an intercontinental deployment model poses additional significant reduction in throughput. Unfortunately the log-linear model here fails to provide an accurate percentage approximation, however the empirical results show system runtime increases of over 3500% on average.

OLS Regression Results

Dep. Variable:	RuntimeFactor	R-squared:	0.925
Model:	OLS	Adj. R-squared:	0.920
Method:	Least Squares	F-statistic:	170.7
Date:	Mon, 16 Mar 2020	Prob (F-statistic):	2.59e-30
Time:	18:44:48	Log-Likelihood:	-0.86006
No. Observations:	60	AIC:	11.72
Df Residuals:	55	BIC:	22.19
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.7200	0.105	16.433	0.000	1.510	1.930
IsMultiNode[T.True]	-0.2493	0.094	-2.663	0.010	-0.437	-0.062
IsMultiDC[T.True]	0.2167	0.094	2.314	0.024	0.029	0.404
IsMultiContinent[T.True]	1.9180	0.094	20.488	0.000	1.730	2.106
Workers	-0.0040	0.002	-1.974	0.053	-0.008	6.19e-05

Omnibus:	28.502	Durbin-Watson:	1.646
Prob(Omnibus):	0.000	Jarque-Bera (JB):	95.623
Skew:	1.226	Prob(JB):	1.72e-21
Kurtosis:	8.678	Cond. No.	167.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Figure 1
OLS regression output.

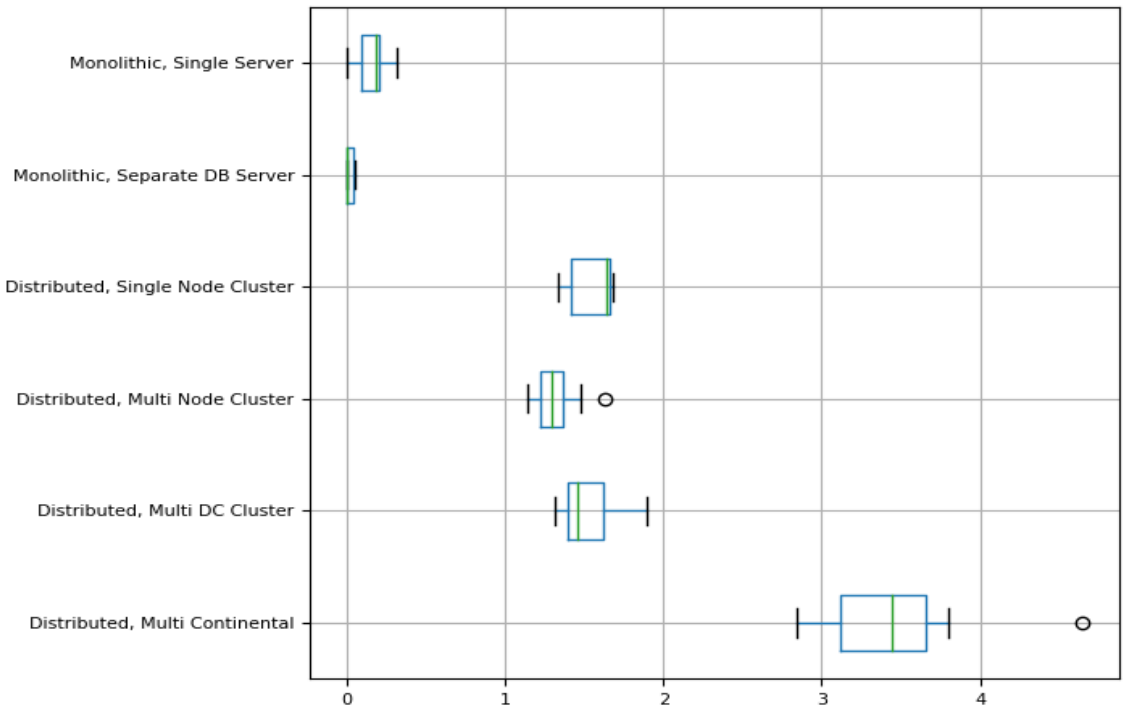

Figure 2
Boxplot showing the percentage increase in runtime, grouped by Batch Description.

Summary and Future Work

This research finds that the performance cost of moving from an ACID compliant monolithic RDBMS to an ACID compliant distributed database is material. Software development teams should expect at least a 50% reduction in write throughput when deploying a multi-node database cluster in a close geographic area. Expanding to an intercontinental cluster reduces write throughput even more, which requires further study for mitigating the costs of distribution. Increasing the number of concurrent connections does have a mitigating impact, however the magnitude of concurrency is not large enough to combat the effects of intercontinental redundancy. Further, clusters spread across multiple data centers realize performance losses directly related to the total network distance, and further work should be explored on options to provide redundancy within smaller geographic networks when appropriate.

There are several opportunities for further work including varying the workloads (perhaps adhering to a standard such as TPC), considering the challenges of a geographically distributed user set, varying the parameters tested within data center choice and total network distance, and evaluating redundancy through the choice of multiple service providers instead of geographic expansion.

Appendix

i. Software system - “main.go” - https://github.com/mcgraw-bb25/ddb-cost-model/blob/master/main.go
ii. Collected data - “master_spreadsheet.csv”, “master_spreadsheet.ods” - https://github.com/mcgraw-bb25/ddb-cost-model/blob/master/master_spreadsheet.csv
iii. Regression output - “regression.py”, “regression.log” - https://github.com/mcgraw-bb25/ddb-cost-model/blob/master/regression.py https://github.com/mcgraw-bb25/ddb-cost-model/blob/master/regression.log
iv. Plot output - “plots.py” - https://github.com/mcgraw-bb25/ddb-cost-model/blob/master/plots.py
v. Example initialization scripts - “init.sh”, “pg1.sh”, “pg2.sh”, “cr1.sh”, “cr2.sh”, “cr3.sh”, “cr4.sh” https://github.com/mcgraw-bb25/ddb-cost-model/blob/master/init.sh https://github.com/mcgraw-bb25/ddb-cost-model/blob/master/pg1.example.sh https://github.com/mcgraw-bb25/ddb-cost-model/blob/master/cr1.example.sh
vi. Lahman Database CSV - “batting.csv”, “pitching.csv”, “fielding.csv” - https://github.com/chadwickbureau/baseballdatabank/archive/master.zip

References

i IBM Research Labs (Chamberlin et al), “A History and Evaluation of System R”, Communications of the ACM, October 1981, Volume 24, Number 10

ii Derrick Harris, GigaOm, December 2011, https://gigaom.com/2011/12/06/facebook-shares-some-secrets-on-making-mysql-scale/

iii James Hamilton, Perspectives, August 2008, https://perspectives.mvdirona.com/2008/05/petascale-sql-db-at-yahoo/

iv Kyle Kingsbury, Jepsen: MongoDB (2.4.3), May 2013 https://aphyr.com/posts/284-call-me-maybe-mongodb

v USENIX ODSI, October 2012 Spanner: Google’s Globally-Distributed Database

vi Thompson et al, ACM SIGMOD, May 2012, “Calvin: Fast Distributed Transactions for Partitioned Database Systems”

vii Cockroach Labs, https://www.cockroachlabs.com/

viii Cockroach Labs, Documentation V19.2, “Life of a Distributed Transaction”

ix Kyle Kingsbury, “CockroachDB beta-20160829”, February2017, https://jepsen.io/analyses/cockroachdb-beta-20160829

x Nikola and Jarke, IEEE Transactions On Knowledge and Data Engineering, Vol 12, No. 4, July/August 2000, “Performance Modeling of Distributed and Replicated Databases”

xi Grim, University of Amsterdam Undergraduate Thesis, June 2016, “Consistency analysis of CockroachDB under fault injection”

xii Kaur and Sachdeva, IEEE International Conference on Inventive Systems and Control, 2017, “Performance Evaluation of NewSQL Databases”