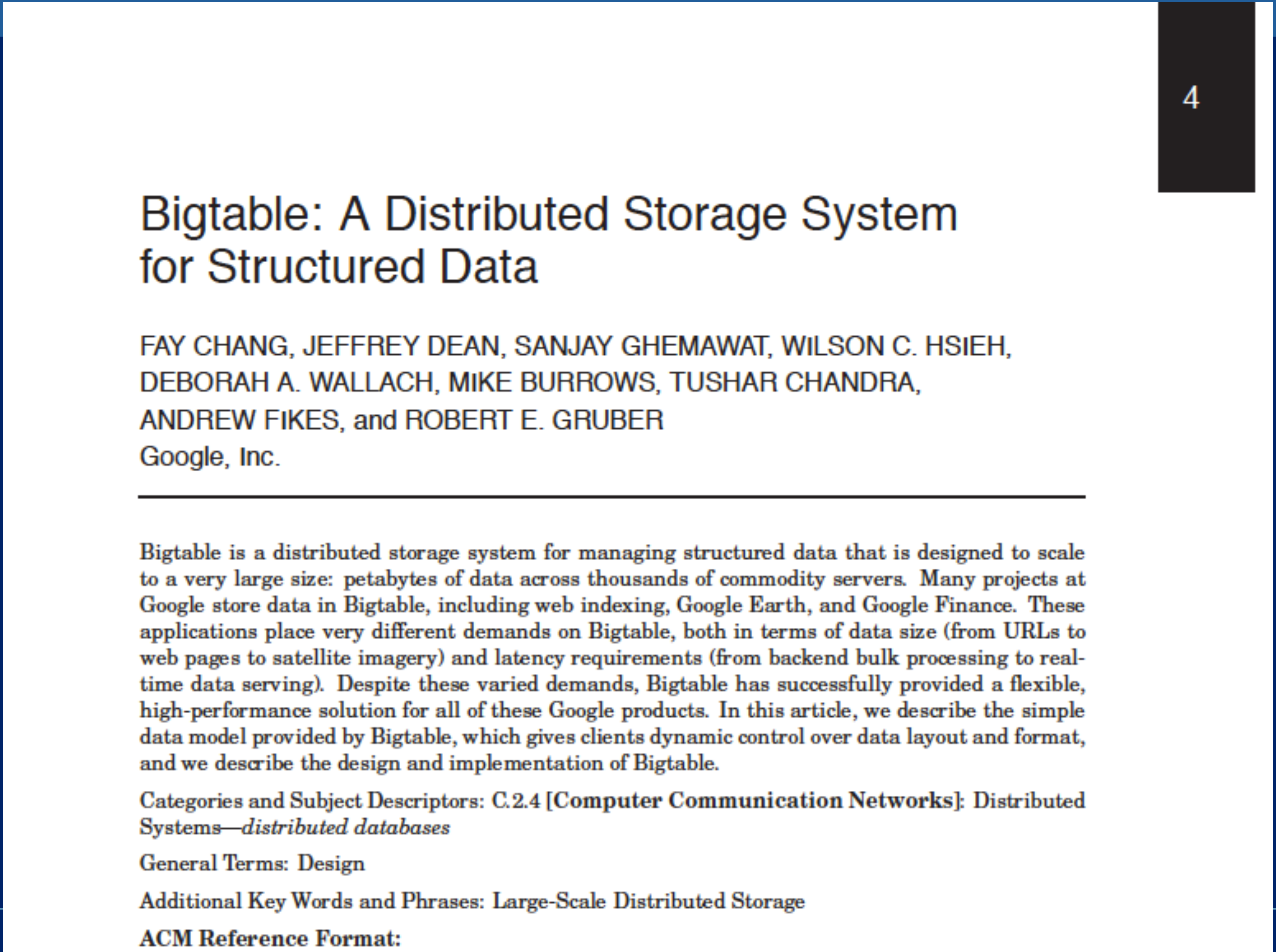
BigTable

BigTable

ACM Trans. Computer Sys., 26(2):A4, June 2008.



4

Bigtable: A Distributed Storage System for Structured Data

FAY CHANG, JEFFREY DEAN, SANJAY GHEMAWAT, WILSON C. HSIEH,
DEBORAH A. WALLACH, MIKE BURROWS, TUSHAR CHANDRA,
ANDREW FIKES, and ROBERT E. GRUBER
Google, Inc.

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this article, we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

Categories and Subject Descriptors: C.2.4 [**Computer Communication Networks**]: Distributed Systems—*distributed databases*

General Terms: Design

Additional Key Words and Phrases: Large-Scale Distributed Storage

ACM Reference Format:

BigTable: Overview

BigTable manages large-scale structured data; designed to reliably scale to petabytes of data spread across thousands of machines.

BigTable is widely applicable within Google, used by >60 Google products and projects; it is scalable, high-performance, and high-availability.

Workload types vary from batch processing jobs where data-throughput rates are key, to serving data to end-users where latency is to be minimised.

BigTable runs on clusters ranging in size "…from a handful to thousands of servers, and store up to several hundred terabytes of data".

BigTable is *the* seminal "NoSQL" database: ultra-large-scale but not fully relational. Data indexed using just row and column names, arbitrary strings.

Data is treated as uninterpreted strings (allows serialized structures).

Locality of data, and whether it is to be served from memory or from disk, is under control of the client – important for latency.

BigTable: Data Model

A BigTable cluster is a set of processes running BigTable.

Each cluster serves a set of tables.

Each table is a sparse, distributed, persistent, sorted map.

Map is from 3 dimensions (row, column, time) onto a string value (a “cell”).

Rows: data is sorted in alpha-order by row key – strings <64Kb long.

Rows with consecutive keys are grouped into tablets: c.200Mb

Columns: a table can have any number of columns. Column keys are grouped into sets called column families, units of access control.

Timestamps: different cells can contain multiple versions of the same data, differentiated by timestamp (a 64-bit integer, microsecond resolution).

BigTable garbage collection can keep most recent *n* versions of data, and/or keep only data that is “within *n* seconds of now” (i.e., retain-recent).

BigTable: API

BigTable API gives access to functions for: put/get of table entries; creating/deleting tables and column families; & for changing metadata (such as access control rights) associated with clusters, tables, and column families.

Clients can write/delete values in BigTable, retrieve row values, and/or iterate over subsets of data in a table.

Single-row transactions are supported: data stored under a single row key can be subject to atomic (all-or-nothing) read-modify-write sequences.

Cells can be used as integer counters.

Clients can provide scripts that are executed on BigTable servers: scripting language called Sawzall; client scripts can't write back into BigTable but they can produce filtered, summarized, and transformed data from the table.

BigTable can be an input source to; or an output target from, MapReduce.

BigTable read/write access to disk is via GFS.

BigTable Building Blocks: SSTables

BigTable uses GFS to store and log data files.

Also internally uses sorted-string table (SSTable) immutable file format to store BigTable data. Public domain info on BigTable use of SSTable is limited, but Google fellows Dean & Ghemawat made a GitHub open-source release of LevelDB (Ruby), believed to be very similar in its use of SSTables.

Nice explanation by Ilya Grigroik (Google engineer) here:

https://www.igvita.com/2012/02/06/sstable-and-log-structured-storage-leveldb/

Dropping down another layer of detail, the way in which manipulations of SSTables is split between working on them in-memory and on-disk requires familiarity with a data structure called a *log-structured merge tree*, coverage of which is simply beyond the scope of this 10-minute mini-lecture.

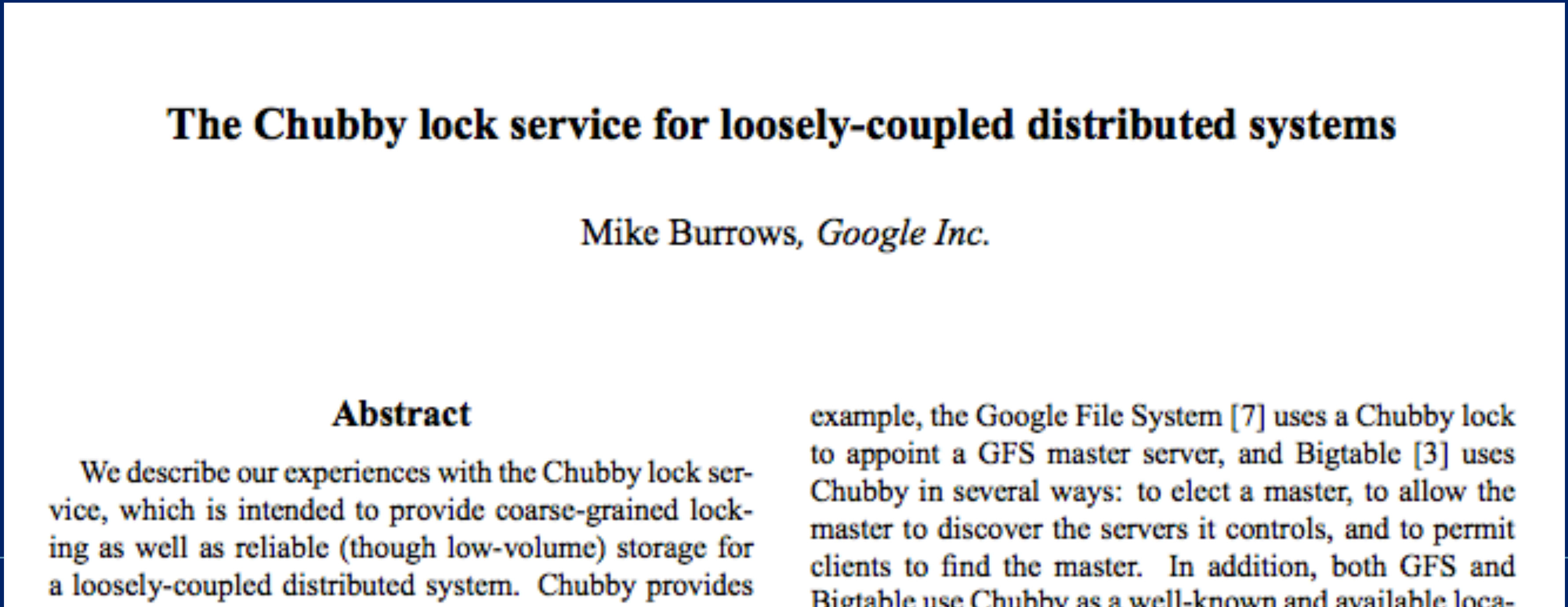
P. O'Neil, E. Cheng, D. Gawlick *et al.* (1996), The Log-Structured Merge Tree (LSM-Tree). *Acta Informatica,* 33(4):351-385.

BigTable Building Blocks: Chubby/Paxos

BigTable also uses Chubby distributed lock service for shared-resource access synchronization, a system that prioritises availability and reliability rather than high performance – implements Paxos (Lecture07).

Google engineer Mike Burrows has a nice 16-page paper on experiences of using Chubby here:

http://static.googleusercontent.com/media/research.google.com/en//archive/chubby-osdi06.pdf



The Chubby lock service for loosely-coupled distributed systems

Mike Burrows, *Google Inc.*

Abstract

We describe our experiences with the Chubby lock service, which is intended to provide coarse-grained locking as well as reliable (though low-volume) storage for a loosely-coupled distributed system. Chubby provides

example, the Google File System [7] uses a Chubby lock to appoint a GFS master server, and Bigtable [3] uses Chubby in several ways: to elect a master, to allow the master to discover the servers it controls, and to permit clients to find the master. In addition, both GFS and Bigtable use Chubby as a well-known and available loca-

Why care about BigTable?

Apache HBase (on HDFS) is directly inspired by BigTable (on GFS).

Apache Cassandra builds on BigTable's data model, with fully distributed (peer-to-peer) design inspired by Dynamo (cf. Amazon DynamoDB).

G. Decandia, D. Hastorun, M. Jampani *et al.* (2007), Dynamo: Amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review,* 41(6):205-220.

Google's Dean & Ghemawat open-source "cousin" of BigTable, LevelDB has been ported to Android, Linux/Unix, MacOSX, and Windows.

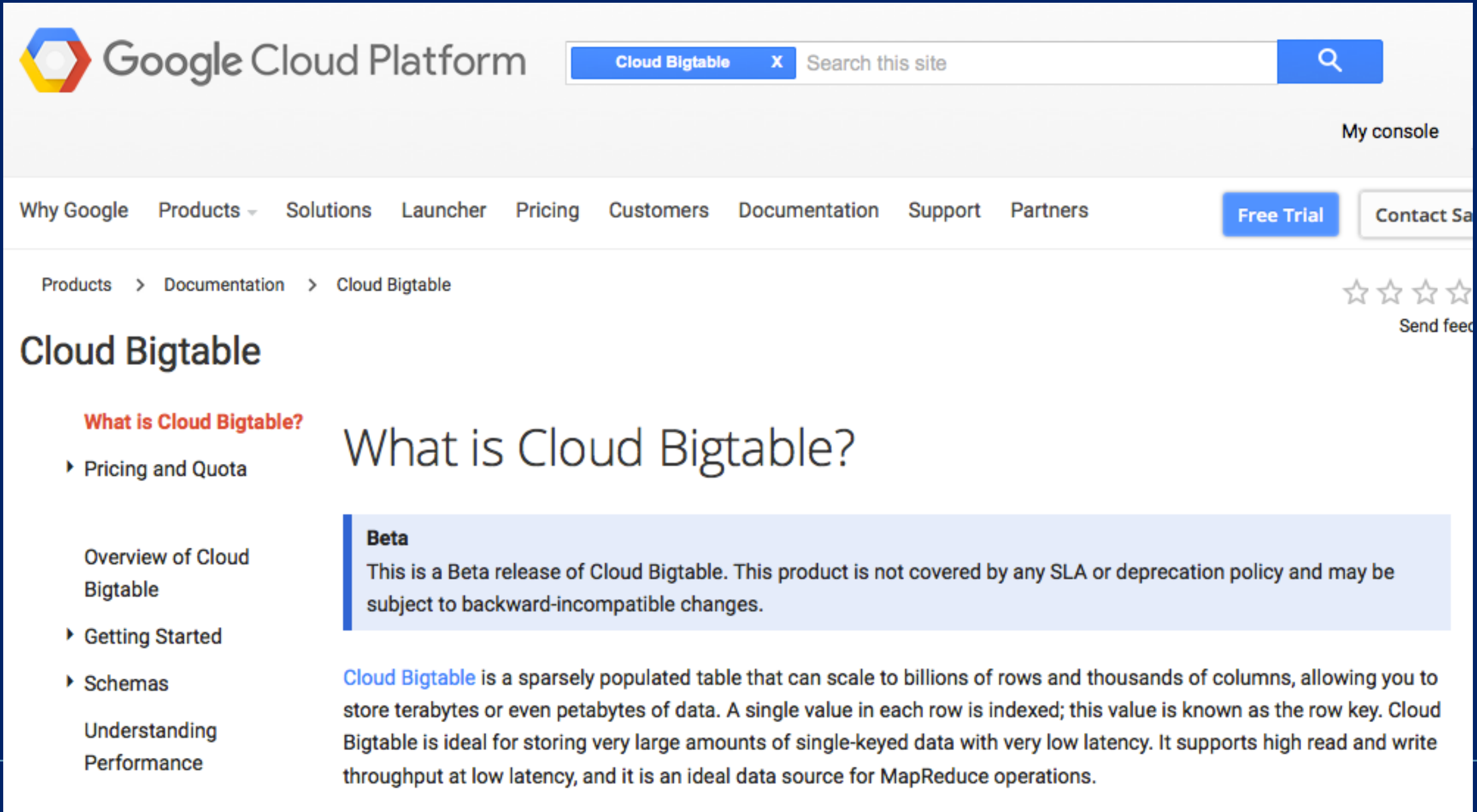
Google Chrome's IndexedDB uses LevelDB as backend database.

Reliability of LevelDB has been questioned: poor if underlying file system doesn't offer atomic & order-preserving operations.

May 2015: The circle closes…

Google Cloud Platform launches Google Cloud BigTable (shakey beta).

GCBT is compatible with HBase/Cassandra, but advertised as much much faster, cheaper, and directly integratable with rest of Cloud Platform.



Cloud Bigtable

What is Cloud Bigtable?

Beta

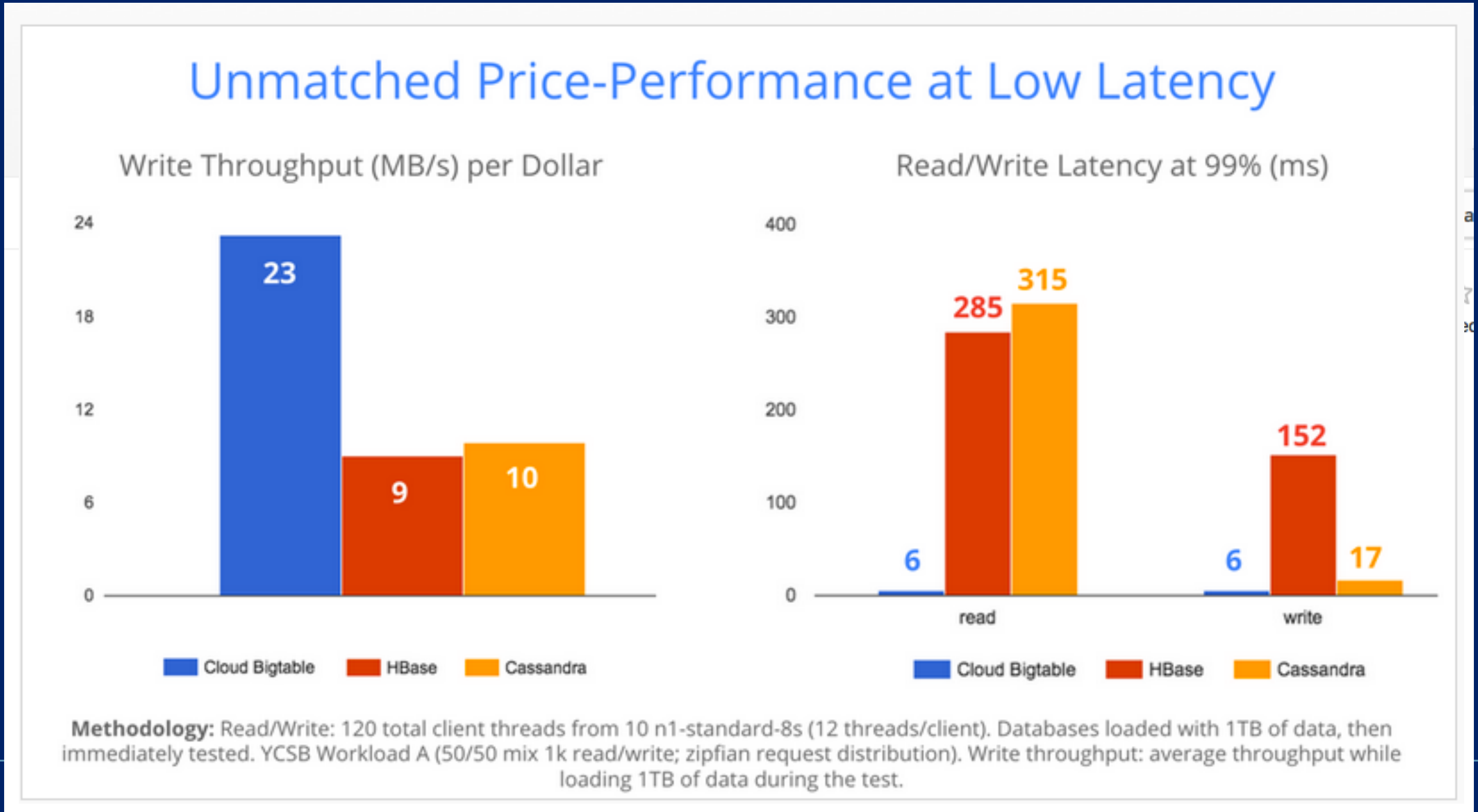
This is a Beta release of Cloud Bigtable. This product is not covered by any SLA or deprecation policy and may be subject to backward-incompatible changes.

Cloud Bigtable is a sparsely populated table that can scale to billions of rows and thousands of columns, allowing you to store terabytes or even petabytes of data. A single value in each row is indexed; this value is known as the row key. Cloud Bigtable is ideal for storing very large amounts of single-keyed data with very low latency. It supports high read and write throughput at low latency, and it is an ideal data source for MapReduce operations.

May 2015: The circle closes…

Google Cloud Platform launches Google Cloud BigTable (shakey beta).

GCBT is compatible with HBase/Cassandra, but advertised as much much faster, cheaper, and directly integratable with rest of Cloud Platform.



Unmatched Price-Performance at Low Latency

Methodology: Read/Write: 120 total client threads from 10 n1-standard-8s (12 threads/client). Databases loaded with 1TB of data, then immediately tested. YCSB Workload A (50/50 mix 1k read/write; zipfian request distribution). Write throughput: average throughput while loading 1TB of data during the test.

Spanner