Distributed Systems: Google App Engine 1

Stefan Walraven, Wouter Joosen

iMinds-DistriNet, KU Leuven

Overview

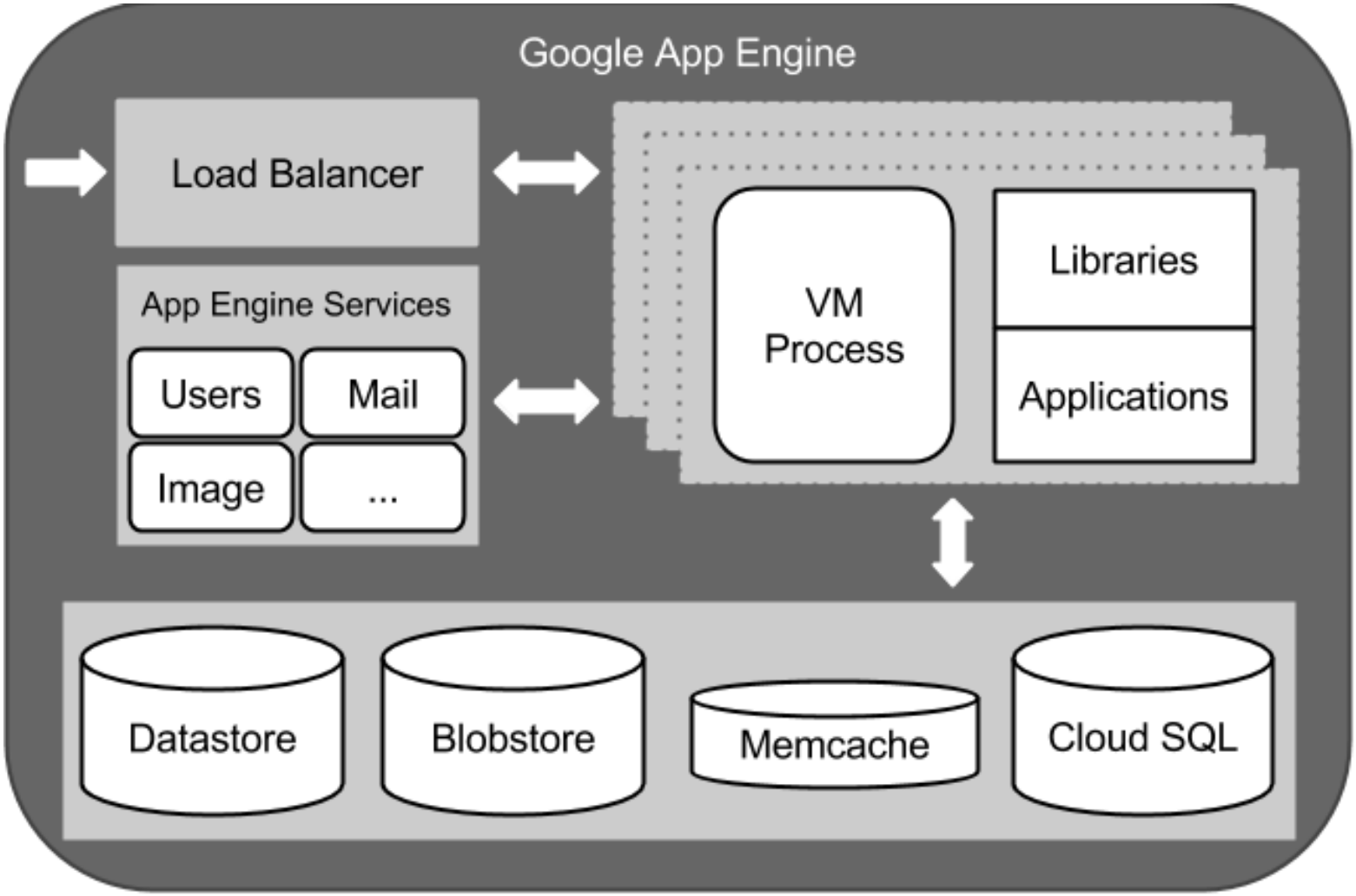
- Introduction
  - What is Google App Engine?
  - Architecture
  - Tool support
- App Engine Technologies
- App Engine Datastore
  - Overview
  - ⇔ JPA in Java EE sessions
  - Transactions

Introduction to App Engine

What is Google App Engine?

- **Platform-as-a-Service (PaaS) platform**
- Targeted at traditional *web applications*
  - Enforcing clean separation between a stateless computation tier and a stateful storage tier
  - Less suitable for general-purpose computing
- *Automatic* scaling and load balancing
- Optimized for applications with *short-lived requests*
  - Within seconds or will not scale!
- Free developer account
  - Only pay for resources used above the free quotas

GAE Architecture



GAE Architecture

- The runtime environments are custom built to ensure that applications run **quickly, securely, and without interference** from other applications (i.e. sandbox)
  - Clones are automatically created when needed (scalability)
- Variety of common services (e.g. users and mail services)
- Data services:
  - Distributed datastore: schemaless object storage (NoSQL datastore)
  - Blobstore: binary large objects (blobs), cf. Amazon S3
  - Memcache: distributed in-memory key-value storage (for caching)
  - Cloud SQL: fully managed, highly available relational databases (MySQL-based)

Tool Support

- Google App Engine SDK
  - Local development server and storage
  - Simulates the GAE environment
    - Not completely equal!
  - Support for writing unit tests with dependencies to GAE services
- Plugin for Eclipse (Java EE version)
  - Create, test and upload applications

App Engine Technologies

Programming languages, restrictions, and differences with Java EE

GAE Technologies
Programming languages

- Supported programming languages:
  - **Python** (not our focus)
    - Including the Python standard library
  - **Go** (not our focus)
    - A fast, statically typed, compiled language
    - for CPU-intensive and distributed tasks
  - **Java (→ presentation and lab sessions)**
    - Java SE 7
    - common web application technologies
    - any other language using a JVM-based interpreter or compiler (e.g. JavaScript, Ruby or Scala)
  - **PHP** (not our focus)
- Extensible with third-party libraries (*if supported by platform*)

GAE Technologies
Restrictions

- Sandbox environment:
  - Limited access to the underlying platform and operating system
  - Isolates application in own secure, reliable environment
- **Restrictions:**
  - Only accessible via HTTP/HTTPS
  - *White list* of allowed library features (JRE classes)
    - E.g. no sockets, no writing to files, limited use of multi-threading
  - Only access to external services through the provided URL fetch and email services
  - Request-reply based
    - only in response to a web request, a queued task, or a scheduled task
    - limited CPU time for a particular request (60s)

GAE Technologies
Differences with Java EE

- **Java EE**
  - Focus on multi-tier enterprise applications
    - Enterprise Java Beans (EJBs)
    - Persistence (JPA)
    - Web technologies: Java Servlets, Server Pages, Web Services (SOAP & REST)...
  - Distributed transactions
- **GAE**
  - Focus on scalable web applications
    - No business tier => **No** support for Enterprise Java Beans (EJB)
    - Persistence: JPA and JDO (but **limited!**)
    - Web technologies: Java Servlets, Java Server Pages
      - Limited support for web services
  - Limited transaction support

GAE Technologies
Differences with Java EE

- Focus of this presentation:
  - **persistence and transactions**

- Where do these differences come from?
  - Java EE: **SQL database** (Java DB)

  - GAE: **NoSQL storage system** (App Engine Datastore)

GAE Technologies
Differences with Java EE

- Structured storage (NoSQL)
  - ⇔ relational database management systems (RDBMS)
  - No fixed tables: *schemaless*
  - *Scales horizontally* ⇔ vertical scaling (relational databases)
    - Scale by adding nodes ⇔ scale by adding resources to single node
    - Suitable for distributed context, such as cloud computing
      - Elasticity!
  - Often *weak consistency* guarantees (no ACID), for example:
    - Eventual consistency (after some time, all replicas will be consistent)
    - Only transactions for single data items

App Engine Datastore

Based on Megastore

Megastore: overview

- **Megastore**
  - Complex **distributed storage system** designed by Google for *interactive online services*:
    - Email, Social networking (Wave, Buzz, G+) and chat (Google Hangouts)
    - Collaborative documents (Google Docs)
    - Maps
    - **Google App Engine (GAE) applications**
    - ...
  - Requirements:
    - Highly scalable
    - Highly available (resilient to failures)
    - Low latency (=> responsive)
    - Consistent view of data
    - Rapid development

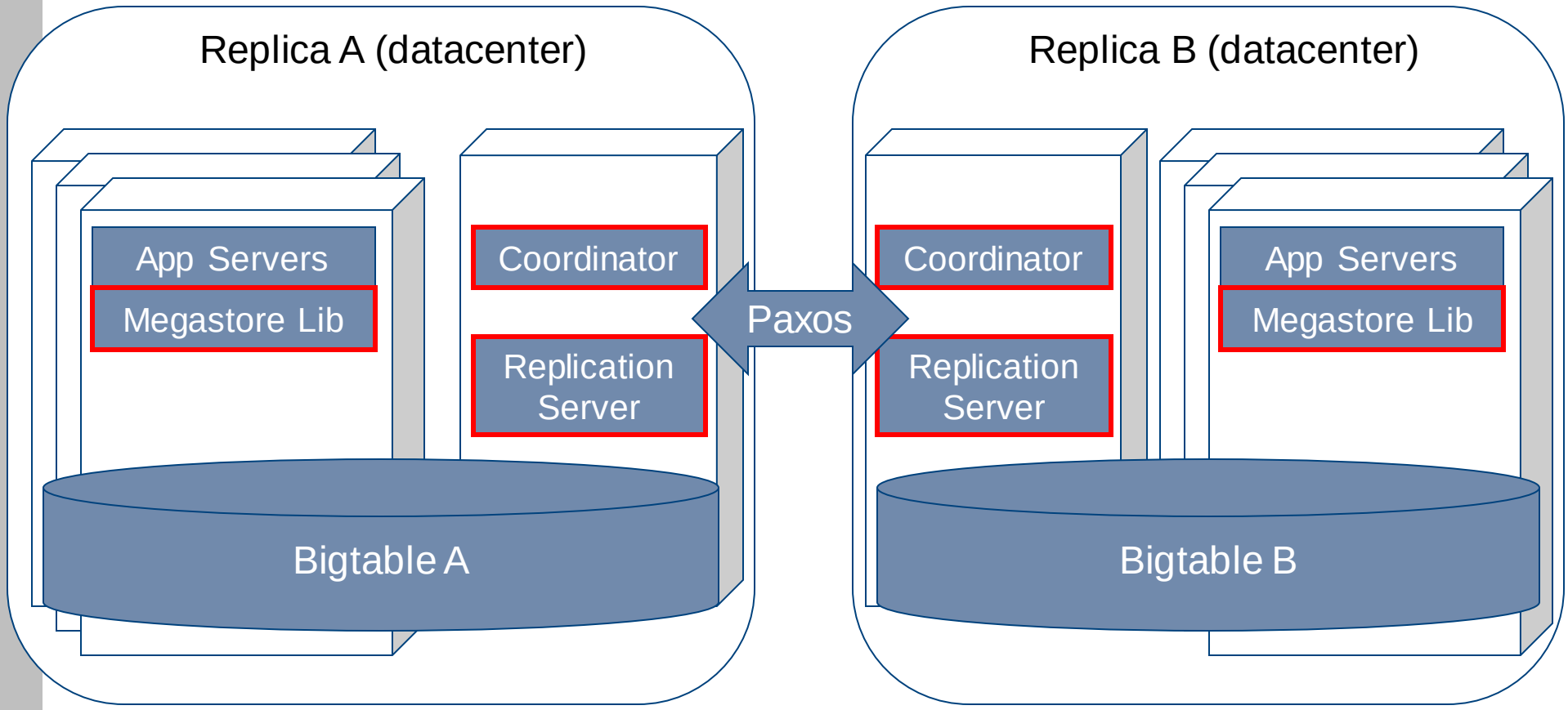
Megastore: overview (2)

- Combination of
  - **Scalability of NoSQL datastore**
    - Partition datastore
    - Replicate partitions separately
    - Full ACID semantics within partitions
    - Limited consistency across partitions
  - **Convenience of traditional RDBMS**
    - Traditional database features (e.g. secondary indexes)
    - BUT ONLY:
      - if features scale within user-tolerable latency limits
      - if semantics are supported by partitioning scheme

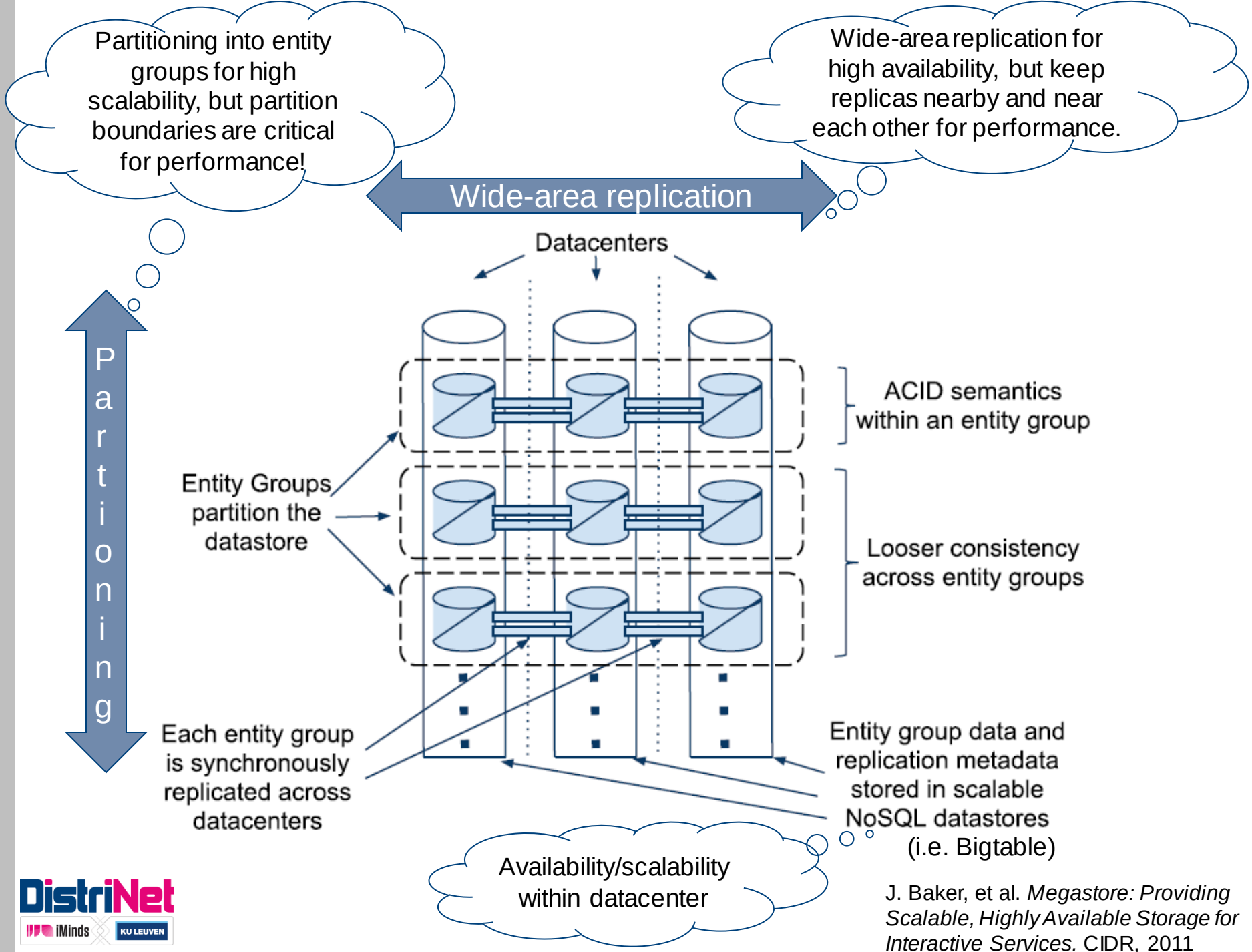
Megastore: overview (3)

- Heavily relies on
  - **Bigtable** (within datacenter)
    - **Distributed storage system for managing structured data** that is designed to scale to a very large size
    - Sparse, multi-dimensional sorted map
      - (row:string, column:string, time:int64) → string
      - **Atomic read/write per row**
      - Unbounded amount of columns per table, grouped in families
      - Timestamps for versioning of cells
  - Paxos (across datacenters)
    - Family of **protocols to reach consensus** in a distributed context
  - Chubby
    - a **lock service** for loosely-coupled distributed systems to synchronize access to shared resources

Megastore: Architectural overview



Note: Coordinators and Replication Servers can run on separate nodes, or on the same nodes as application servers (i.e. clients). But there is only one coordinator and replication server per replica.



J. Baker, et al. *Megastore: Providing Scalable, Highly Available Storage for Interactive Services.* CIDR, 2011

Megastore: data model

- Schema
  - Strongly typed
  - Contains set of tables
- Data represented by ***entities***
  - Contain named and typed properties (i.e. fields)
  - Primary key: sequence of properties (unique in table)
  - Mapped to a single Bigtable row
    - Primary key => Bigtable row key
    - Other properties => own Bigtable column
  - **But:** entities of the same kind can have a different schema

Megastore: data model (2)

- **Hierarchical persistence model**
  - Each entity group consists of:
    - **1 root entity**
    - Many child entities that refer to root entity
      - Part of primary key of child entity
  - Visible in data schema
    - Child tables declare foreign key referencing entity group root table
  - ⇔ Relational persistence model (RDBMS)

Megastore: data model (3)

- **Strong consistency within entity group**
  - Transaction log for entity group stored into root entity row
    - Atomic access via single Bigtable transaction
    - Replicated => any node can initiate read/write
  - Use Paxos to achieve consensus for appending entry to transaction log
- **Looser consistency across entity groups**
  - Using indirect communication
  - Optional: 2-phase commit (very expensive!)

Megastore in GAE

- Provides object-relational mapping from RDBMS to NoSQL
- Support for **Java Persistence API (JPA) (→ lab sessions)**
  - Standard interface for RDBMS, cf. Java EE
- Support for Java Data Objects (JDO)
  - agnostic to the technology of the underlying datastore
  - data objects are ordinary POJOs
- **BUT differences and limitations…**

JPA on App Engine Datastore ⇔ JPA in Java EE

- Google-specific rules for primary keys
  - Primary key includes: application ID, kind, and entity ID
  - Key for **root of entity group:**
    - Long
    - String
    - Key
    - Key as encoded String
  - Key for **child entities:**
    - Key
      - Includes the key of the entity group parent
    - Key as encoded String
  - Generation type for primary keys should be IDENTITY

(com.google.appengine.api.datastore.Key)

JPA on App Engine Datastore ⇔ JPA in Java EE

- App Engine:

```
@Entity
public class Address {
   @Id
   @GeneratedValue(strategy=
         GenerationType.IDENTITY)
   private Key id;
   // other properties
   ...
}
```

- Java EE

```
@Entity
public class Address {
   @Id
   @GeneratedValue(strategy=
         GenerationType.AUTO)
   private Long id;
   // other properties
   ...
}
```

JPA on App Engine Datastore ⇔ JPA in Java EE (2)

- All queries are served by pre-built indexes
- Therefore, **types of queries are restricted**:
  - No many-to-many relations
  - No “Join” queries
  - No aggregation queries (group by, having, sum, avg, max, min)
  - No polymorphic queries
    - You only get entities of the specific type you queried for (no subclasses)
- More advanced JPQL queries from Java EE will not work any more

JPA on App Engine Datastore ⇔ JPA in Java EE (3)

- No container-managed entity manager:
  - No @PersistenceContext
  - Create singleton EntityManagerFactory
    - Time-consuming operation => reuse
  - **Manually** create and close EntityManager instance when needed
    - **Be aware of lazy loading!**

JPA on App Engine Datastore ⇔ JPA in Java EE (3)

- App Engine:

```
EntityManager em =
  EMF.get().createEntityManager();
try {
      // ... do stuff with em ...
   } finally {
      em.close();
   }
```

- Java EE:

```
@PersistenceContext
EntityManager em;

// ... do stuff with em ...
```

JPA on App Engine Datastore ⇔ JPA in Java EE (4)

- Transactions (optional):
  - Restriction on what can be done inside a *single* transaction:
    - Operate on entities in the **same entity group**
    - Single transaction cannot create or operate on more than one root entity
  - *Optimistic concurrency*
    - Multiple concurrent transactions changing the same entity group
    - First transaction to commit will succeed, all others will fail
  - *Cross-Group (XG) transactions* supported
    - Perform queries on separate entity groups (**max 5!**)
    - Higher latency
    - Extra property in persistence.xml

```
<property name="datanucleus.appengine.datastoreEnableXGTransactions" value="true"/>
```

JPA on App Engine Datastore ⇔ JPA in Java EE (4)

- No Java EE/JTA support in GAE
  - No container-managed (global) transactions
  - No knowledge about entity groups

```
EntityTransaction tx = em.getTransaction();
tx.begin();
try {
   // do something, e.g. em.persist()
   tx.commit();
} finally {
   if (tx.isActive())
       tx.rollback();
}
```

Links

- Getting started
  https://cloud.google.com/appengine/docs/java/gettingstarted
- Will it play in App Engine?
  http://code.google.com/p/googleappengine/wiki/WillItPlayInJava
- F. Chang, et al. *BigTable: A Distributed Storage System for Structured Data*. OSDI, 2006
  http://research.google.com/archive/bigtable-osdi06.pdf
  (http://static.usenix.org/event/osdi06/tech/chang/chang_html/)
- J. Baker, et al. *Megastore: Providing Scalable, Highly Available Storage for Interactive Services.* CIDR, 2011
  http://research.google.com/pubs/archive/36971.pdf
- Datastore Overview
  https://cloud.google.com/appengine/docs/java/datastore/
- **Using JPA with App Engine** 
  https://cloud.google.com/appengine/docs/java/datastore/jpa/overview-dn2#Setting_Up_JPA_2_0