



AWS Summit

AWS技术峰会 2015 · 上海





DynamoDB最佳实践

Jenny Sun 孙素梅

AWS解决方案架构师，区域主管



议程

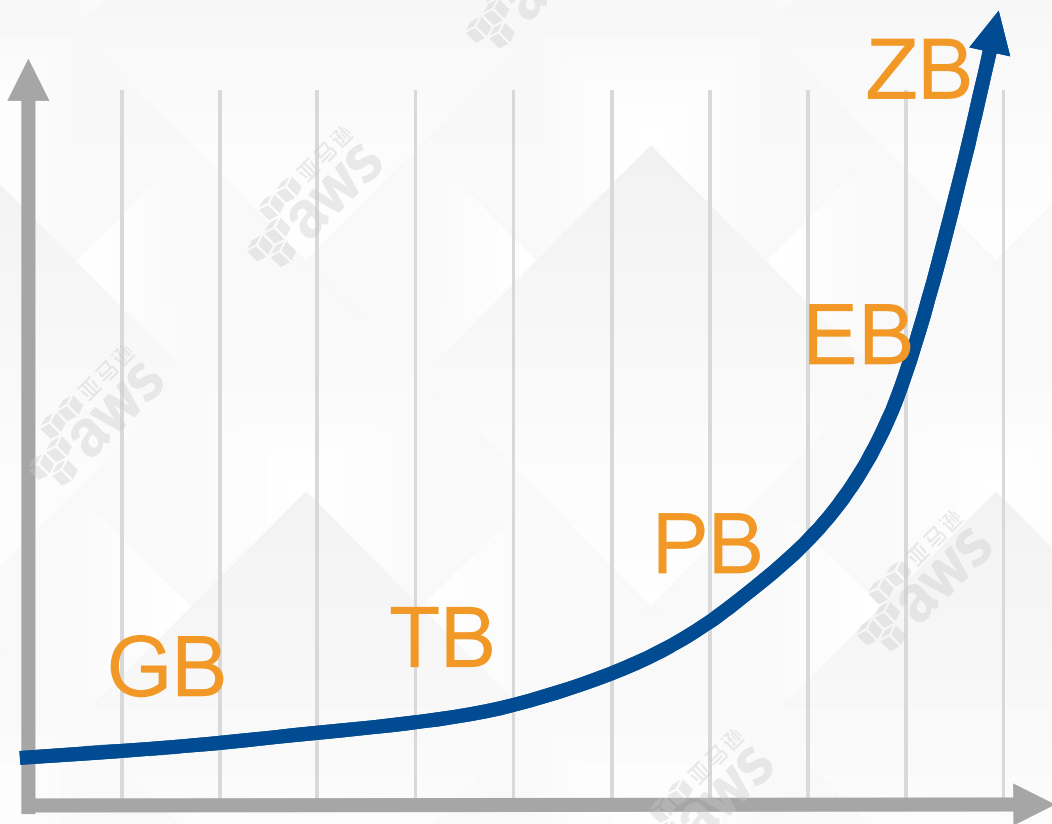
- 为什么选择DynamoDB
- 什么是DynamoDB
- 基本知识
- 数据模型
- 应用场景和最佳实践
- 总结



为什么选择DynamoDB



大数据的趋势和特点



- 大数据(3V):
 - 量大
 - 数据增长速度很快: MB/s很常见; GB/s也越来越普遍
 - 多种数据: 日志, 性能监控数据, 用户指标, 安全数据, IoT...
- 互联用户数目惊人速度增长
 - 预计2020年75 billion互联设备
- 10^5 或者 10^6 transactions/s在大数据应用中并不少见

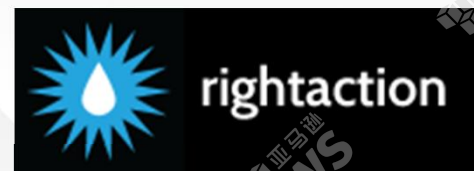
客户需要一个这样的数据库...

- 稳定的可预测的低延迟响应
- 灵活的查询和数据模型
- 支持大规模，能够无限扩展，弹性的吞吐量和存储空间
- 数据强一致性
- 内置高可用和容错能力

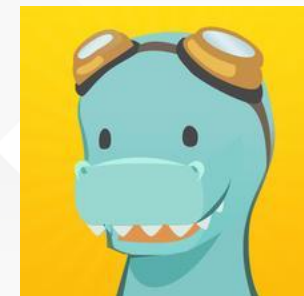
DynamoDB位于大数据的黄金点



DynamoDB客户



互联网时光机Timehop



- 2014年最大表的行数**26亿行**（2,675,812,470）
- 2014年3月至5月的8周内，注册用户从每天5000用户飙升至每天5.5万用户，延时被严格的控制在毫秒级别，通常是**4~6ms**。DynamoDB充当了坚实的后盾
- 2015年4月，最大表的行数飙升到**600亿行**，每天平均发生**1.6亿次写操作**。

“当下我们的表格已经非常大了，接近100TB，但是性能和第一天搭建时没有任何区别。”



什么是DynamoDB



Amazon DynamoDB

- AWS全面管理的NoSQL数据库服务
- 全部基于solid-state drives (SSDs)
- 没有存储空间上限
- 可以支撑任意数量的每秒并发吞吐量
- 稳定的低延迟的性能：单位数ms的响应延迟
- 同时支持Key-Value和Document数据模型
- 自动在三个AZ复制数据
- 低成本

设计理念

- 非常简单的开始使用
- 稳定的可预测的高性能
- 耐久性和可用性
- 用户无需关注数据库的扩展性
- 低成本



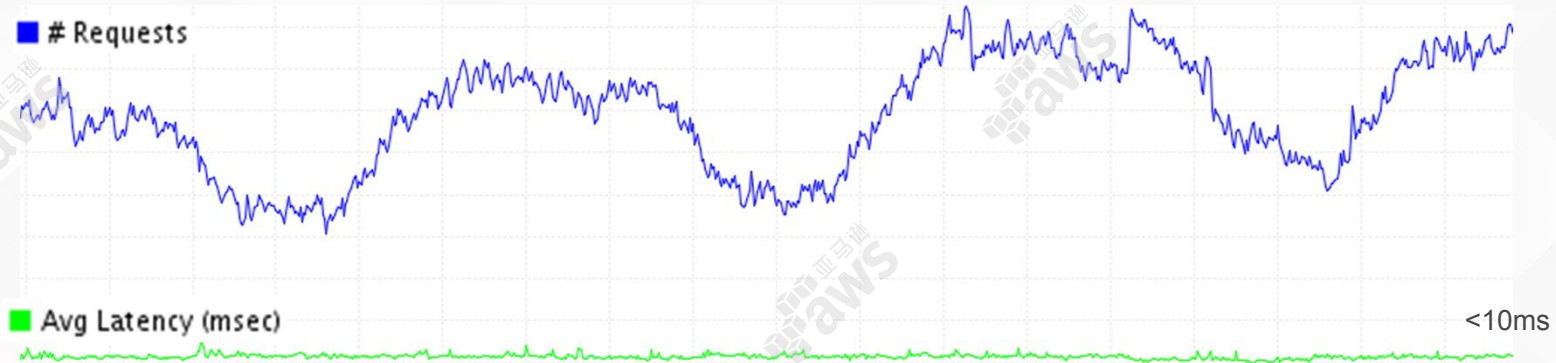
专注于你的业务

设计理念 -- 使用简单

每秒并发吞吐量(RCU/WCU)
主键

做两个决定 + 点几下鼠标
= 可以开始使用

设计理念 -- 稳定的可预测的低延迟响应

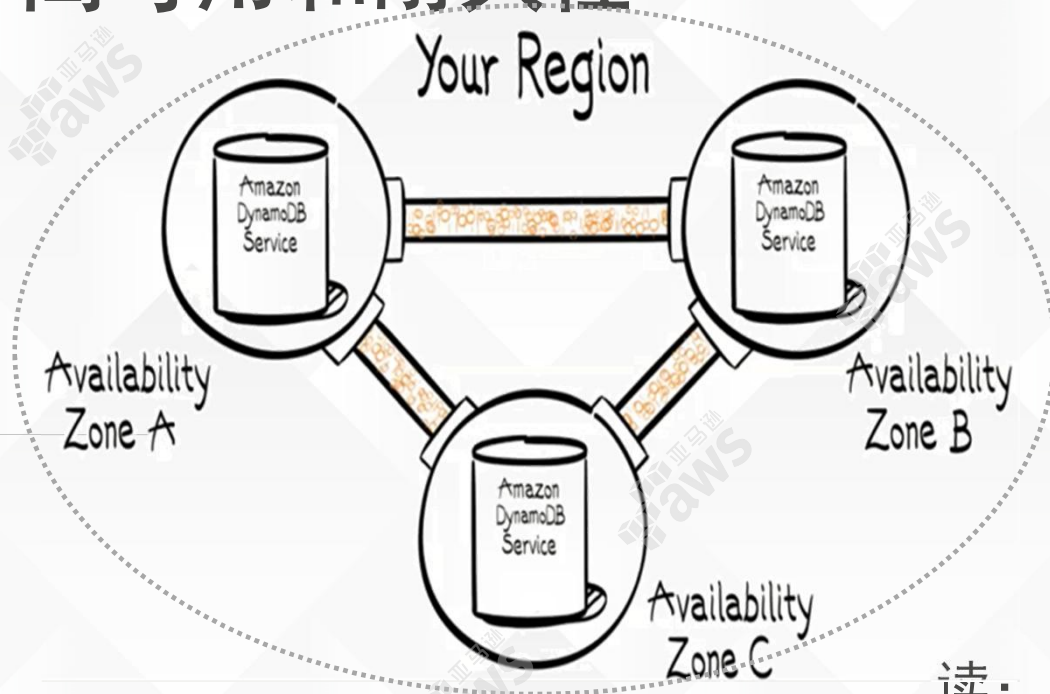


Docs, SDK's: <http://aws.amazon.com/dynamodb/developer-resources/>

Auto Scaling Your DynamoDB: <https://github.com/sebdah/dynamic-dynamodb>



设计理念 -- 高可用和耐久性



写:

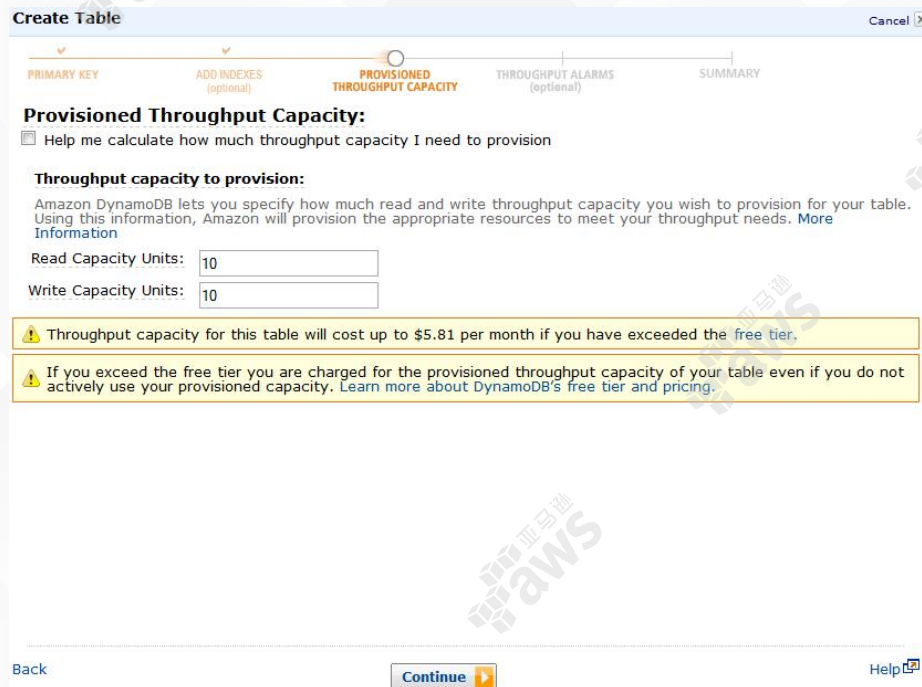
- 自动在三个AZ同步复制数据
- 每个写至少在两个AZ写成功后才会返回
- Disk-only writes

读:

- 支持强一致性和最终一致性读
- 一样的低延迟
- 最终一致性的成本是强一致性成本的一半

设计理念 – 无缝扩展

- 只需要告诉我们你需要的每秒并发吞吐量 (API/Console)
- 剩下的交给AWS: auto-partitioning



The screenshot shows the 'Create Table' wizard in the AWS Management Console. The 'Provisioned Throughput Capacity' step is active, showing options to help calculate capacity or manually specify Read and Write Capacity Units (both set to 10). It includes a warning about costs exceeding the free tier.

Create Table [Cancel]

PRIMARY KEY | ADD INDEXES (optional) | **PROVISIONED THROUGHPUT CAPACITY** | THROUGHPUT ALARMS (optional) | SUMMARY

Provisioned Throughput Capacity:

☐ Help me calculate how much throughput capacity I need to provision

Throughput capacity to provision:

Amazon DynamoDB lets you specify how much read and write throughput capacity you wish to provision for your table. Using this information, Amazon will provision the appropriate resources to meet your throughput needs. [More Information](#)

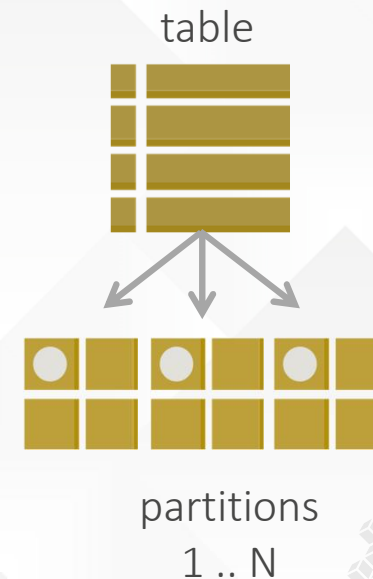
Read Capacity Units:

Write Capacity Units:

⚠ Throughput capacity for this table will cost up to \$5.81 per month if you have exceeded the [free tier](#).

⚠ If you exceed the free tier you are charged for the provisioned throughput capacity of your table even if you do not actively use your provisioned capacity. [Learn more about DynamoDB's free tier and pricing.](#)

[Back](#) [Continue](#) [Help](#)



设计理念 -- 低成本

- 提供free usage tier
- 只需要为预配置的吞吐量和实际使用的存储空间付费
 - \$0.0065/h for 10 WCU (每小时可以支撑多达36,000写)
 - \$0.0065/h for 50 RCU (每小时可以支撑多达180,000强一致性读, 或者360,000最终一致性读)
 - First 25 GB stored per month is free, \$0.25 per GB-month thereafter
- 每天一杯咖啡的价钱(\$4.16)完全可以为10万名玩家提供支持 (<http://blog.csdn.net/awschina/article/details/38562221>)
- 支持预留容量 (有Reserved Capacity) 省得更多
- DynamoDB Local提供线下开发环境, 提供full API支持



DynamoDB基本知识

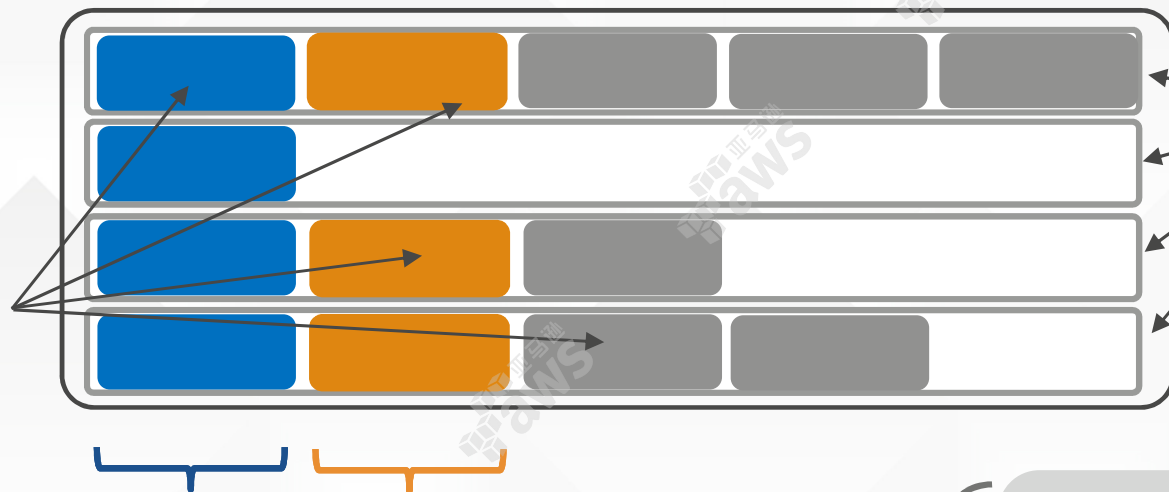


表格，项目，属性

属性

表格

项目



必需
键-值访问模式，查找: {=}
决定数据的分布

可选
1:N 关系模型
支持丰富的查询模型

All items for a hash key
==, <, >, >=, <=
"begins with"
"between"
sorted results
counts
top/bottom N values
paged responses

DynamoDB数据类型

- String (S) *
 - Number (N) *
 - Binary (B) *
 - String Set (SS)
 - Number Set (NS)
 - Binary Set (BS)
 - Boolean (BOOL)
 - Null (NULL)
 - List (L)
 - Map (M)
- 用于存储JSON文档

*可以被用作Hash Key或者Range Key的数据类型

预配置的吞吐量

- 读容量单位 (RCU: Read Capacity Units)
 - 1 RCU = 4KB 强一致性读
 - 1 RCU = 8KB 最终一致性读
- 写容量单位 (WCU: Write Capacity Units)
 - 1 WCU = 1KB 写
- 每项目操作（包括Batch API）
 - 每个Item，读取按4KB取整，写入按1KB
- 查询和扫描
 - 总量大小，读取按4KB取整，写入按1KB

DynamoDB API

- CreateTable
- UpdateTable
- DeleteTable
- DescribeTable
- ListTables
- GetItem
- Query
- Scan
- BatchGetItem
- PutItem
- UpdateItem
- DeleteItem
- BatchWriteItem



DynamoDB

In preview
Stream API

- ListStreams
- DescribeStream
- GetShardIterator
- GetRecords

扩展和分区

- DynamoDB自动扩展分区
 - 可以配置任意数值的吞吐量，无上限限制
 - 可以存储任意数量的项目，每个项目最大400KB
- 扩展和分区
 - 根据数据存储大小进行扩展：表格的总存储空间；10GB/分区
 - 根据读写吞吐量进行扩展：预配置的读写吞吐量
 - 3000 RCU: 1 分区(partition)
 - 1000 WCU: 1 分区(partition)

计算分区

$$\begin{array}{l} \# \text{ 分区数目} \\ \text{(吞吐量)} \end{array} = \frac{RCU_{for\ reads}}{3000\ RCU} + \frac{WCU_{for\ writes}}{1000\ WCU}$$

$$\begin{array}{l} \# \text{ 分区数目} \\ \text{(表格大小)} \end{array} = \frac{Table\ Size\ in\ GB}{10\ GB}$$

$$\# \text{ 总分区数目} = \text{MAX}(\begin{array}{l} \# \text{ 分区数目} \\ \text{(表格大小)} \end{array} \mid \begin{array}{l} \# \text{ 分区数目} \\ \text{(吞吐量)} \end{array})$$

将来，也可能会发生变化...

分区和吞吐量

- 每个分区预留相同的吞吐量
- 一张拥有3个分区的表格，如果预配置的WCU是1500，则会平均分配到每个分区中
- **留意表的历史配置:** 避免over provisioning

Item #1	#2	...				
Partition #1			Partition #2		Partition #3	
500 WCU			500 WCU		500 WCU	

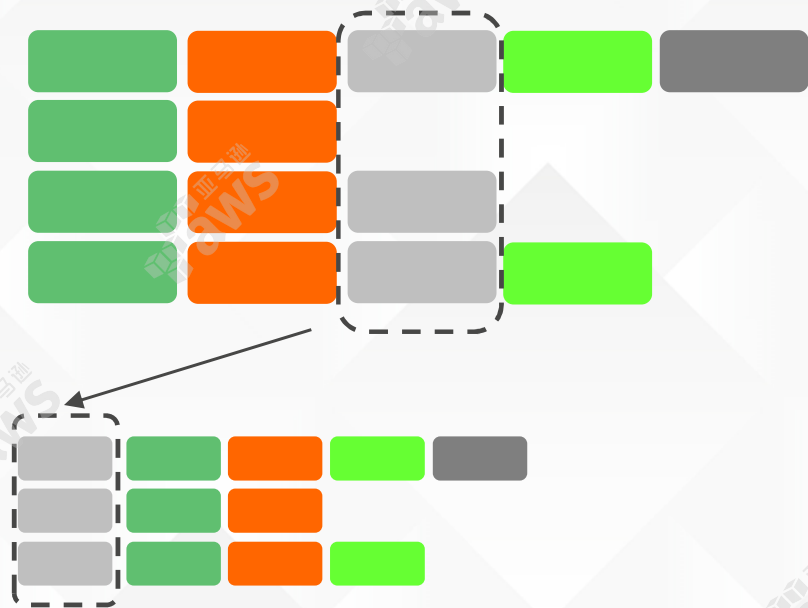
预配置吞吐量

项目和分区

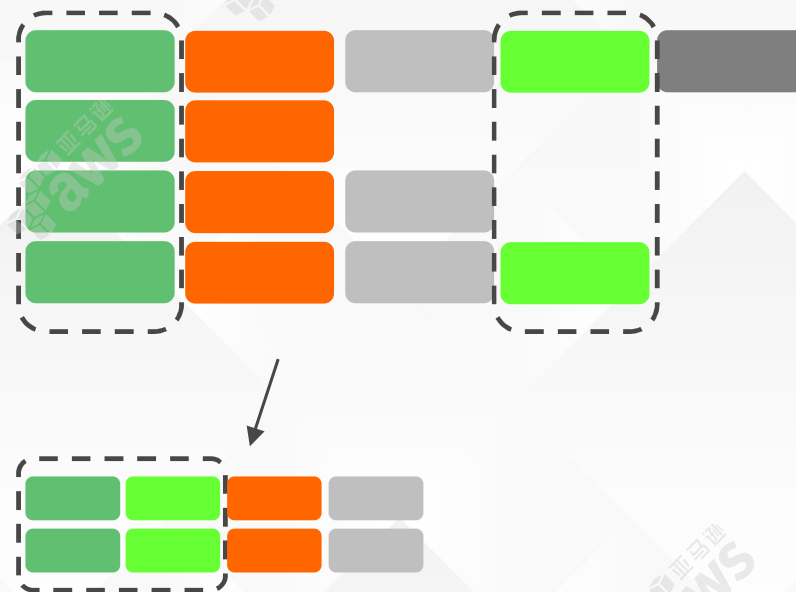
- 一个项目，一个分区
- 项目按照hash key被放置到不同分区中



Index选项 – GSI 和 LSI



全局二级索引(GSI)
选择任意属性做新的hash
key和range key



本地二级索引(LSI)
相同的hash key+不同的range key
索引数据和表格数据位于相同分区

二级索引对比和选择

- 全局二级索引(GSIs)
 - 最终一致性
 - 无限扩展
 - 可以选择任意hash key
 - 支持动态创建和删除索引
 - 独立的预配置吞吐量
- 本地二级索引 (LSIs)
 - 强一致性
 - 最多10GB数据
 - Hash key是base table的Hash key
 - 在创建表格的时候指定
 - 和base table共享吞吐量

如果一个项目的数据量多于10GB，选择GSI

如果业务场景接受数据最终一致性，选择GSI



数据模型

1:1关系 或者 key-values

- 设计表格或者GSI时候采用Hash key做主键
- 采用GetItem 或者 BatchGetItem API

例子: 指定SSN 或者 license number, 获取属性值

Users Table	
Hash key	Attributes
SSN = 123-45-6789	Email = johndoe@nowhere.com, License = TDL25478134
SSN = 987-65-4321	Email = maryfowler@somewhere.com, License = TDL78309234

Users-Email-GSI	
Hash key	Attributes
License = TDL78309234	Email = maryfowler@somewhere.com, SSN = 987-65-4321
License = TDL25478134	Email = johndoe@nowhere.com, SSN = 123-45-6789

1:N 关系 或者 父-子

- 设计表格或者GSI时采用Hash key + Range Key做主键
- 采用Query API

例子:

指定设备, 查询epoch值在 X, Y之间的所有读数

Device-measurements		
Hash Key	Range key	Attributes
DeviceId = 1	epoch = 5513A97C	Temperature = 30, pressure = 90
DeviceId = 1	epoch = 5513A9DB	Temperature = 30, pressure = 90

N:M 关系

- 表格采用Hash key + Range Key做主键，并选择反转的Hash key + Range Key做GSI
- 采用Query API

例子：

指定用户，查询所有参与的游戏；或者指定游戏，查询所有参与的用户。

User-Games-Table	
Hash Key	Range key
UserId = bob	GameId = Game1
UserId = fred	GameId = Game2
UserId = bob	GameId = Game3

Game-Users-GSI	
Hash Key	Range key
GameId = Game1	UserId = bob
GameId = Game2	UserId = fred
GameId = Game3	UserId = bob

丰富的表达式

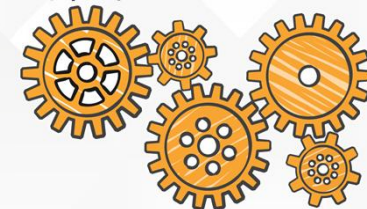
- Projection表达式
 - Query/Get/Scan: 定义获取的属性
- 过滤表达式
 - Query/Scan: $\#V > :num$
- 条件表达式
 - Put/Update/DeleteItem: `attribute_not_exists (#pr.FiveStar)`
- 更新表达式
 - UpdateItem: `set Replies = Replies + :num`



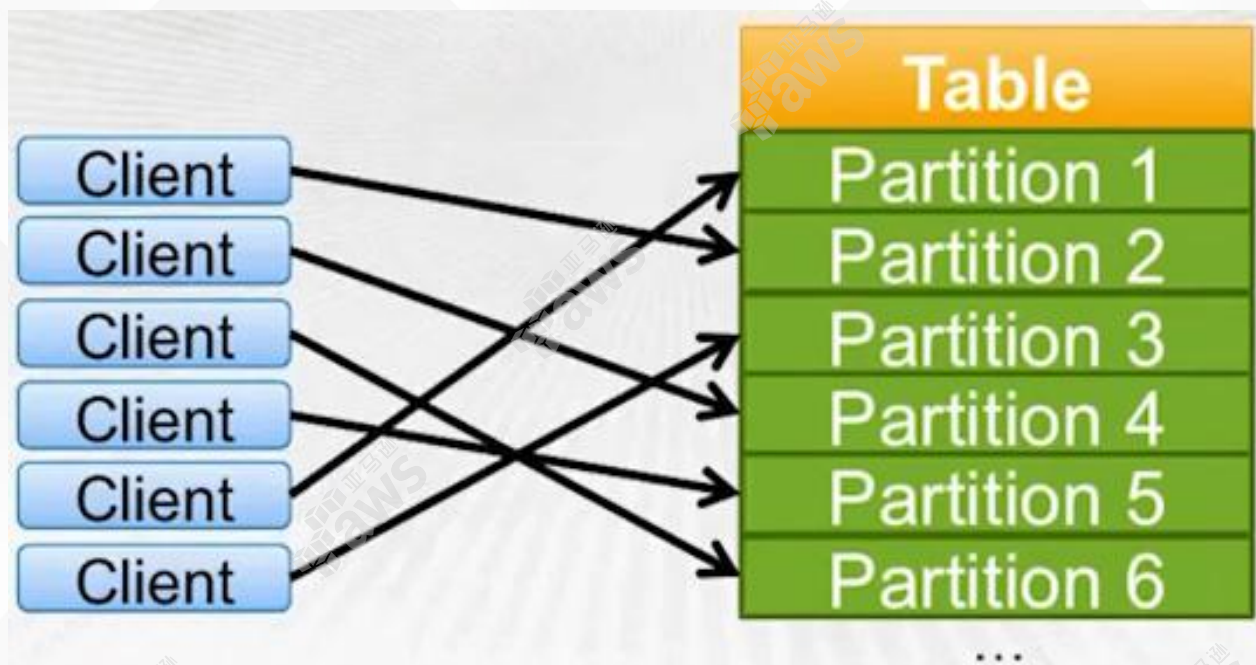
DynamoDB应用场景和最佳实践



最佳实践1：慎重选择Hash Key以实现无限扩展



- 避免hot key
 - 好的hash key取值范围很大；不好的hash key只能有有限的值
 - 选择能将负载均衡分布到不同分区的hash key（访问模式）



好的Hash Key的例子

user_id = mza	first_name = Matt	last_name = Wood
user_id = jeffbarr	first_name = Jeff	last_name = Barr
user_id = werner	first_name = Werner	last_name = Vogels
user_id = simone	first_name = Simone	last_name = Brunozzi
...

大量不同的用户有不同的user_id. 负载能够很好的分布在不同的Hash key也就是不同的partition。

不好的Hash Key的例子

status = 200	date = 2012-04-01-00-00-01
status = 404	date = 2012-04-01-00-00-01
status 404	date = 2012-04-01-00-00-01
status = 404	date = 2012-04-01-00-00-01

Status code取值有限，并且分布是不均匀的，
所以会造成不均衡的工作负载分布

最佳实践2：如何存储大项目

例子：Messaging App



```
SELECT *  
FROM Messages  
WHERE Recipient='David'  
LIMIT 50  
ORDER BY Date DESC
```



Messages App



Messages
Table



David

如果将信息混合存储

Hash key

Range key



Messages Table

<u>Recipient</u>	<u>Date</u>	Sender	Message
David	2014-10-02	Bob	...
... 48 more messages for David ...			
David	2014-10-03	Alice	...
Alice	2014-09-28	Bob	...
Alice	2014-10-01	Carol	...

(Many more messages)

Inbox



David

```
SELECT *  
FROM Messages  
WHERE Recipient='David'  
LIMIT 50  
ORDER BY Date DESC
```

50 items × 256 KB each

大的消息体

计算inbox查询的成本

$$50_{items} \times 256_{KB} \times \frac{1_{RCU}}{4_{KB}} \times \frac{1_{read}}{2_{e.c.reads}} = \mathbf{1600_{RCU}}$$

查询了50条

平均项目大小

Conversion ratio

最终一致性读

把大项目分开存储在另一张表

Uniformly distributes large item reads

(50 sequential items at 128 bytes)

1. Query Inbox-GSI: 1 RCU
2. BatchGetItem Messages: 1600 RCU

(50 separate items at 256 KB)



David

 Inbox-GSI

<u>Recipient</u>	<u>Date</u>	Sender	Subject	MsgId
David	2014-10-02	Bob	Hi!...	afed
David	2014-10-03	Alice	RE: The...	3kf8
Alice	2014-09-28	Bob	FW: Ok...	9d2b
Alice	2014-10-01	Carol	Hi!...	ct7r

 Messages Table

<u>MsgId</u>	Body
9d2b	...
3kf8	...
ct7r	...
afed	...

把大项目存储在S3

1. Query Inbox-GSI: 1 RCU
2. Application fetch items from S3

 Inbox-GSI

Recipient	Date	Sender	Subject	MsgId
David	2014-10-02	Bob	Hi!...	s3://post_msgs/id_100
David	2014-10-03	Alice	RE: The...	s3://post_msgs/id_102
Alice	2014-09-28	Bob	FW: Ok...	s3://post_msgs/id_1045
Alice	2014-10-01	Carol	Hi!...	s3://post_msgs/id_102



David



最佳实践2：如何存储大项目



- Item的大小有限制，但是Item的数目无限制
 - 将一个大的属性分开存储在分开的表格中
 - 将大的条目分成多个小条目中，采用不同的Hash键，形成Virtual Item
 - 将image/media等大文件存储在S3

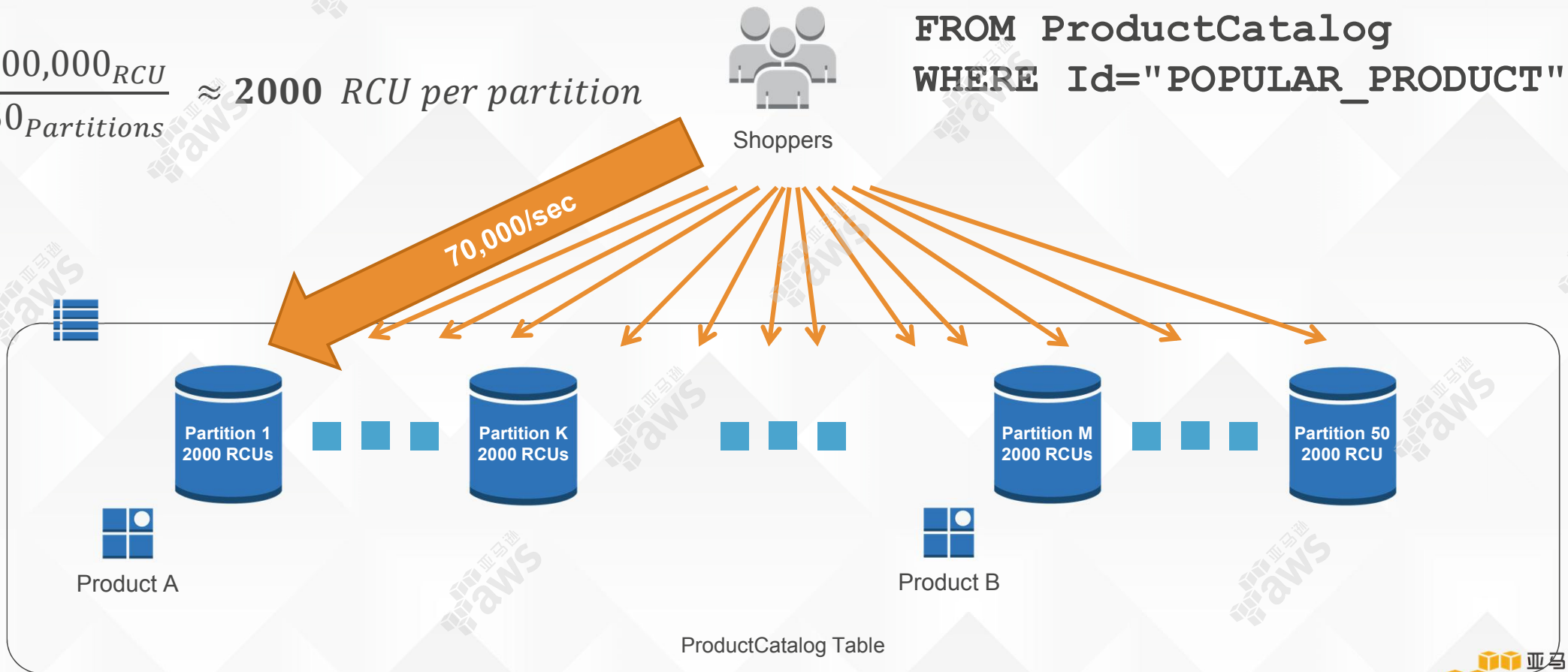
→ 将查询分散到不同分区，降低对单一partition的读写吞吐量，从而降低整表的吞吐量要求

最佳实践3：如何处理热点项目

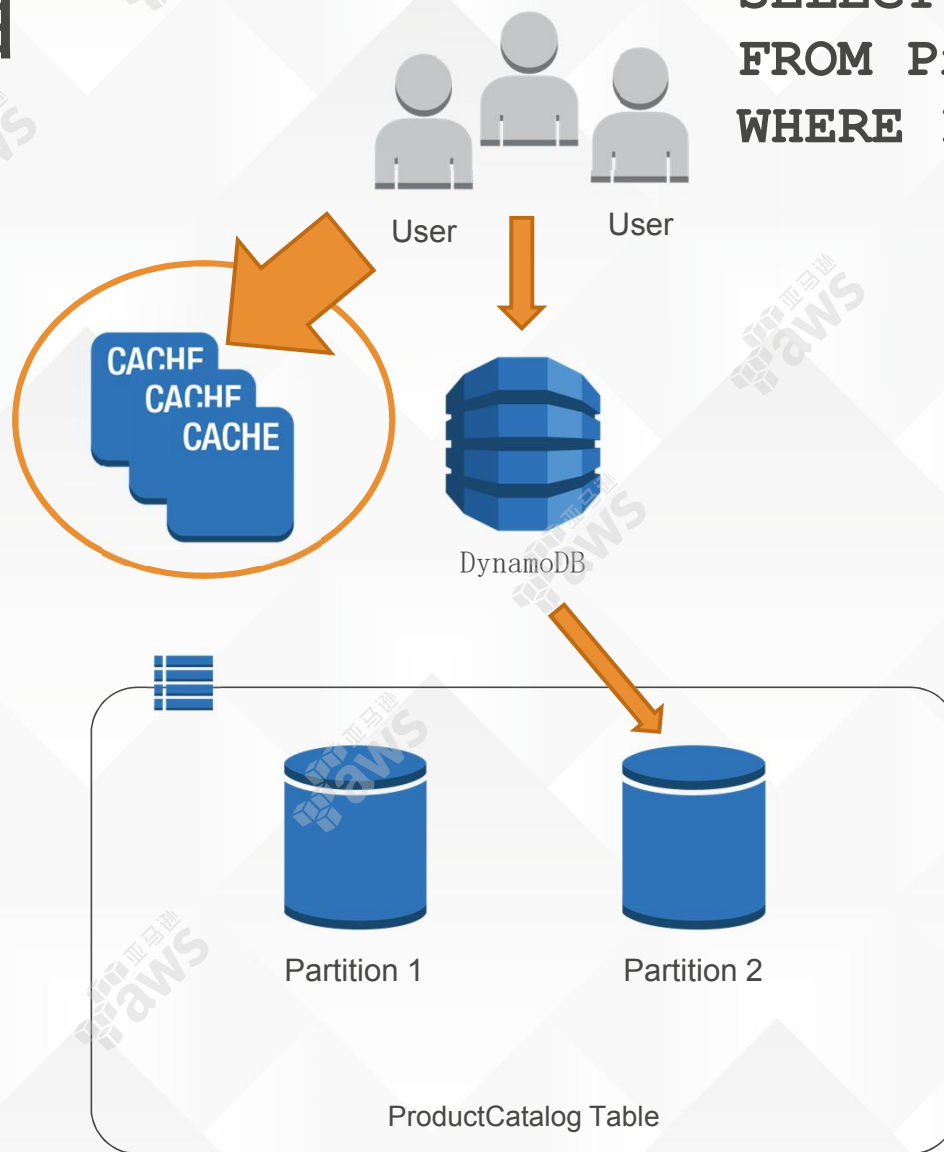
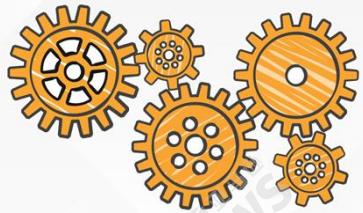
例子：产品目录

$$\frac{100,000_{RCU}}{50_{Partitions}} \approx 2000 \text{ RCU per partition}$$

```
SELECT Id, Description, ...  
FROM ProductCatalog  
WHERE Id="POPULAR_PRODUCT"
```



缓存热点项目



```
SELECT Id, Description, ...  
FROM ProductCatalog  
WHERE Id="POPULAR_PRODUCT"
```

最佳实践4：使用Time-Series表格存储时序型数据

- 适应于类似日志，点击行为，浏览行为，交易记录，操作记录等时序性数据
- 对于应用倾向于访问最近的数据，而老的数据访问很少或者根本不再访问的场景，应该将热数据和冷数据分别存储在不同表格中，这样就可以对热数据表格配置高的吞吐量，冷数据表格配置低的吞吐量
- 预先创建每天，每周或者每月的表格，对当前表格配置需要的吞吐量，新的数据写入当前表格，关闭或者降低历史数据表格的吞吐量

对每个时间段创建一张表



例子：事件日志

当前表格

Events table 2015 April				
<u>Event id</u> (Hash key)	<u>Timestamp</u> (range key)	Attribute1	Attribute N

RCUs = 10000
WCUs = 10000

Hot data

较早的表格

Events table 2015 March				
<u>Event id</u> (Hash key)	<u>Timestamp</u> (range key)	Attribute1	Attribute N

RCUs = 1000
WCUs = 100

Events_table_2015_February				
<u>Event id</u> (Hash key)	<u>Timestamp</u> (range key)	Attribute1	Attribute N

RCUs = 100
WCUs = 1

Events table 2015 January				
<u>Event id</u> (Hash key)	<u>Timestamp</u> (range key)	Attribute1	Attribute N

RCUs = 10
WCUs = 1

Cold data

不要混合存储热数据和冷数据; 对于不用的冷数据可以归档到S3

最佳实践5：如何处理以多个属性为条件的查询

Hash key

Games Table

Gameld	Date	Host	Opponent	Status
d9bl3	2014-10-02	David	Alice	DONE
72f49	2014-09-30	Alice	Bob	PENDING
o2pnb	2014-10-08	Bob	Carol	IN_PROGRESS
b932s	2014-10-03	Carol	Bob	PENDING
ef9ca	2014-10-03	David	Bob	IN_PROGRESS

如何处理游戏邀请查询？ (?)

(range)

(hash)

```
SELECT * FROM Game
WHERE Opponent='Bob'
AND Status='PENDING'
ORDER BY Date DESC
```

方法1: Query filter

Hash key

Range key



Bob



Secondary Index

<u>Opponent</u>	<u>Date</u>	<u>GameId</u>	Status	Host
Alice	2014-10-02	d9bl3	DONE	David
Carol	2014-10-08	o2pnb	IN_PROGRESS	Bob
Bob	2014-09-30	72f49	PENDING	Alice
Bob	2014-10-03	b932s	PENDING	Carol
Bob	2014-10-03	ef9ca	IN_PROGRESS	David

方法1: Query filter

```
SELECT * FROM Game
WHERE Opponent='Bob'
ORDER BY Date DESC
FILTER ON Status='PENDING'
```



Bob



Secondary Index

<u>Opponent</u>	<u>Date</u>	<u>GameId</u>	Status	Host
Alice	2014-10-02	d9bl3	DONE	David
Carol	2014-10-08	o2pnb	IN_PROGRESS	Bob
Bob	2014-09-30	72f49	PENDING	Alice
Bob	2014-10-03	b932s	PENDING	Carol
Bob	2014-10-03	ef9ca	IN_PROGRESS	David

(filtered out)

方法2: 组合键

Status
DONE
IN_PROGRESS
IN_PROGRESS
PENDING
PENDING

+

Date
2014-10-02
2014-10-08
2014-10-03
2014-10-03
2014-09-30

=

StatusDate
DONE_2014-10-02
IN_PROGRESS_2014-10-08
IN_PROGRESS_2014-10-03
PENDING_2014-09-30
PENDING_2014-10-03

方法2: 组合键

Hash key

Range key



Secondary Index

<u>Opponent</u>	<u>StatusDate</u>	<u>GameId</u>	Host
Alice	DONE_2014-10-02	d9bl3	David
Carol	IN_PROGRESS_2014-10-08	o2pnb	Bob
Bob	IN_PROGRESS_2014-10-03	ef9ca	David
Bob	PENDING_2014-09-30	72f49	Alice
Bob	PENDING_2014-10-03	b932s	Carol

方法2: 组合键

```
SELECT * FROM Game
WHERE Opponent='Bob'
AND StatusDate BEGINS_WITH 'PENDING'
```



Bob



Secondary Index

<u>Opponent</u>	<u>StatusDate</u>	<u>GameId</u>	Host
Alice	DONE_2014-10-02	d9bl3	David
Carol	IN_PROGRESS_2014-10-08	o2pnb	Bob
Bob	IN_PROGRESS_2014-10-03	ef9ca	David
Bob	PENDING_2014-09-30	72f49	Alice
Bob	PENDING_2014-10-03	b932s	Carol

最佳实践5：如何处理以多个属性为条件的查询

- 使用Query filter返回少量数据
 - 简化应用代码
 - 简单的类似SQL语句的表达式：AND, OR, NOT, ()
- 使用组合键代替filter
 - 连接两个属性形成有用的二级索引键



最佳实践6： 监控CloudWatch指标避免请求被限制

- 请求被限制的原因
 - 实际读写容量单位超过预配置的总的读写容量
 - 存在hot key，导致负载不均衡出现hot partition，这些分区的读写容量超过预配置的读写容量
 - 热数据和冷数据混合存储
 - 吞吐量被稀释: 1) 当心表格的历史; 2) 冷热数据混合存储
 - 如果表格存在二级索引，会占用额外的容量
 - 不要依赖burst capacity
- Cloudwatch提供延时、消耗的读/写吞吐单元、报错、throttling等指标
- 及时增加预配置的吞吐量



最佳实践7：使用Reserved Capacity

- 预留基准线容量
- 1年预留：~50% 折扣
- 3年预留：~75% 折扣





总结

总结

- 慎重选择Hash Key以实现无限扩展
- 将大的项目和属性分开存储
- 缓存热点项目
- 使用时间序列表格存储时序型数据
- 使用filter或者组合键应对复杂查询
- 监控CloudWatch指标避免请求被限制
- 使用Reserved Capacity



更多最佳实践参考DynamoDB开发指南: <http://aws.amazon.com/documentation/dynamodb/>

小测验

- 题目1

- 表格大小：8GB
- RCU = 5000
- WCU = 500
- 总分区数目 = ? 个

- 题目2

- 列举出选择GSI/LSI的标准

- 题目3

- 列举出DynamoDB的两项最佳实践



Thank You

