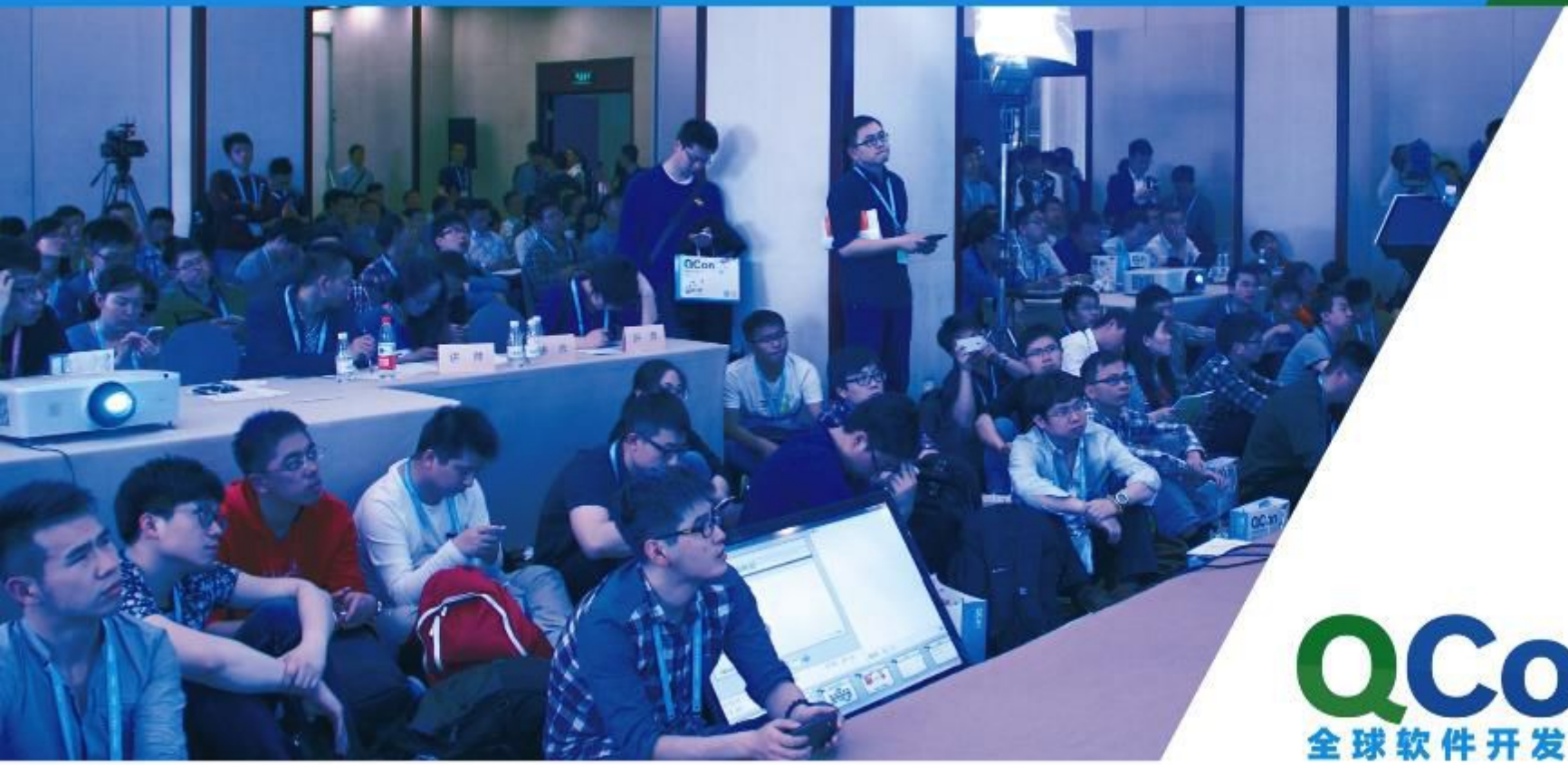
QCon全球软件开发大会

International Software Development Conference



QCon
全球软件开发大会


Geekbang.
极客邦科技
全球领先的技术人学习和交流平台
InfoQ
专注中高端技术
人员的社区媒体
EGO EXTRA GEEKS' ORGANIZATION NETWORKS
高端技术人员
学习型社交网络
StuQ
实践驱动的IT职业
学习和服务平台
扫我，码上开启新世界
Geekbang.
InfoQ | EGO NETWORKS | StuQ

InfoQ

促进软件开发领域知识与创新的传播

高性能数据分析平台架构实践：SequoiaDB+PowerLinux

作者：孔皓

新型分布式数据库

高性能硬件平台加速大数据应用

Spark on PowerLinux之优化方案

数据分析平台案例

新型分布式数据库

数据库领域的挑战者

大数据时代来临拥抱新型数据库Nosql

- 在过去的很长一段时间中，**关系型数据库（Relational Database Management System）**一直是最主流的数据库解决方案，他运用真实世界中事物与关系来解释数据库中抽象的数据架构。然而，在信息技术爆炸式发展的今天，大数据已经成为了继云计算，物联网后新的技术革命，关系型数据库在处理大数据量时已经开始吃力，开发者只能通过不断地优化数据库来解决数据量的问题，但优化毕竟不是一个长期方案，所以人们提出了一种新的数据库解决方案来迎接大数据时代的到来——**NoSQL（非关系型数据库）**。

新一代分布式数据库特点

- 存储模式灵活简单：无需设计表结构和操作模式
- 性能更优：面对大数据的需求，性能优势明显
- 分布式水平扩展：容量动态扩展，大大节约存储成本
- 高可用：在不影响性能的前提下，架构更灵活
- 大数据架构支持：新一代分布式数据库完全的支持Hadoop/Spark等等主流的大数据架构，对于大数据的应用支持更全面

国内新一代分布式数据库代表——SequoiaDB

国内新一代分布式数据库代表——SequoiaDBSequoiaDB（巨杉数据库）是一款分布式文档型NoSQL数据库，是业界唯一支持事务和SQL的产品。SequoiaDB既可作为Hadoop与Spark的数据源以满足实时查询和分析的混合负载，也可独立作为高性能、灵活易用的数据库被应用程序直接使用。

SequoiaDB是大数据应用首选

文档型数据模型

无需设计表和操作模式
更灵活和简单高效

高性能的数据库引擎

提供了业内领先的
读写速度和性能表现

主-从模式的复制组

保证了数据的安全性
实现读写分离

事务功能的支持

SequoiaDB结合二段
提交实现了事务功能

多数据中心复制功能

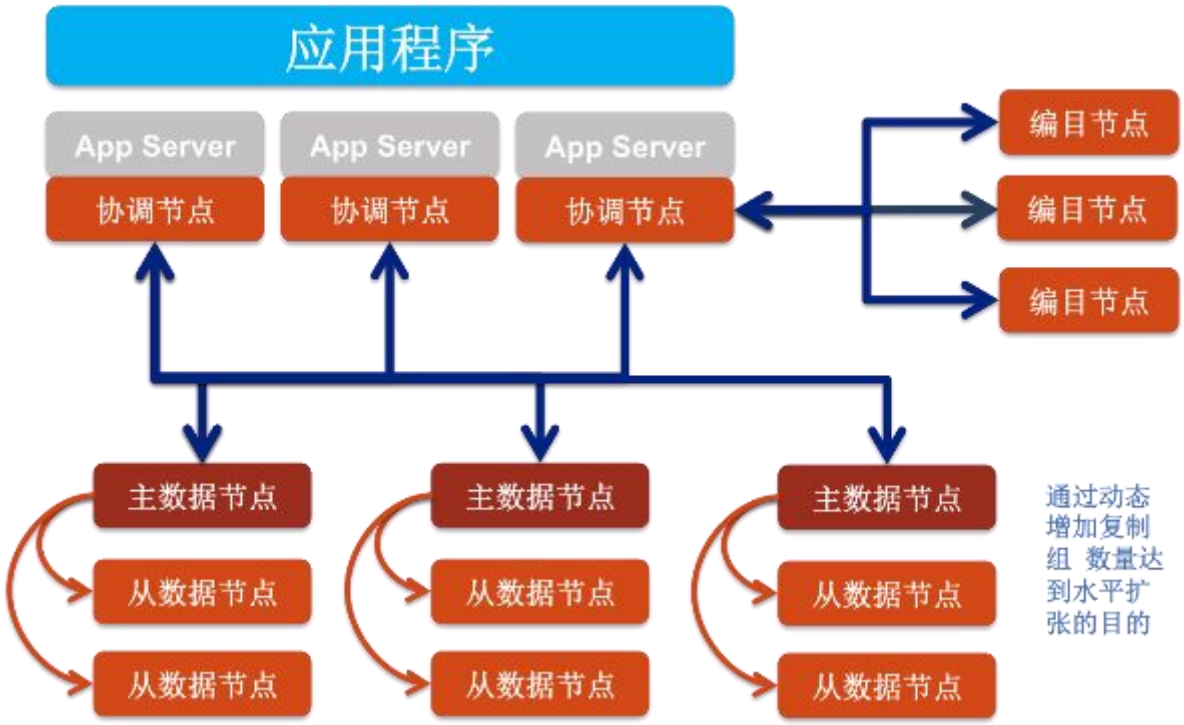
实现了分布式下
灾害备份

多平台架构和语言

为企业提供了
多种开发和运营选择

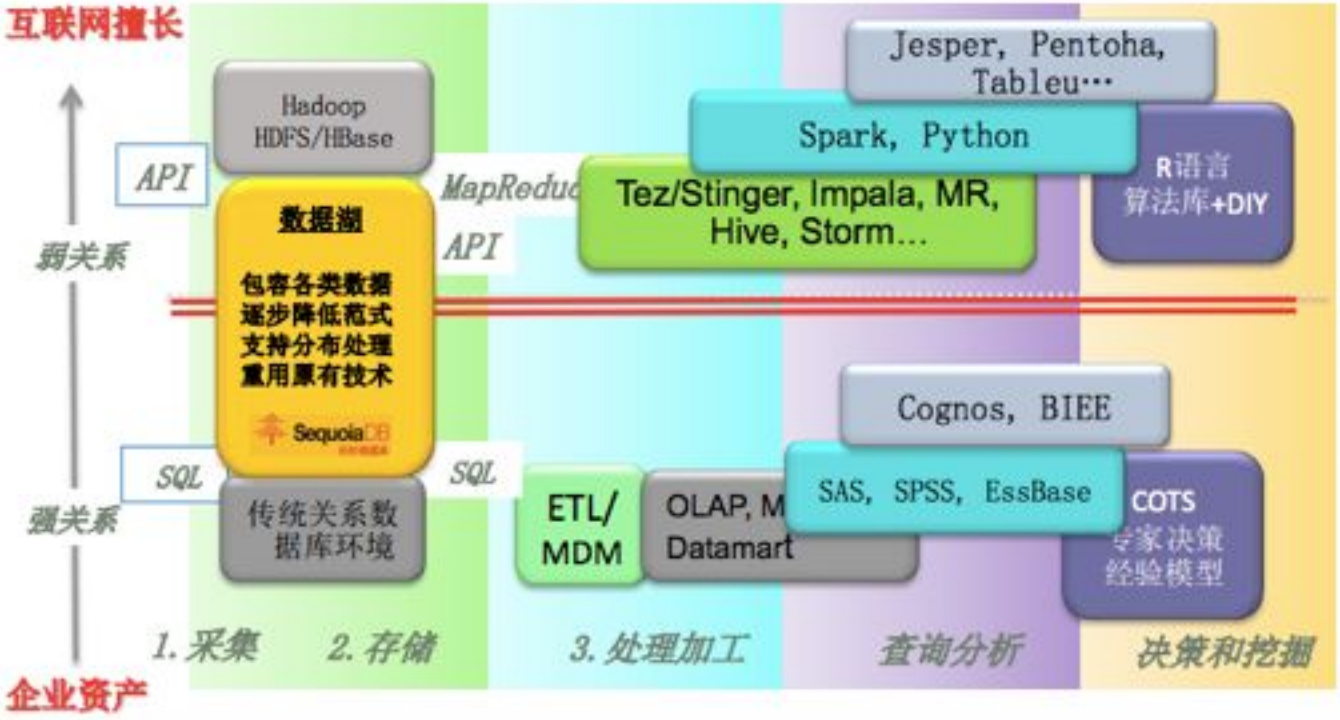
SequoiaDB 存储架构

SequoiaDB目前的架构使用的是典型的MPP架构，编目节点存储元数据，协调节点负责分布式系统的任务分发，数据节点负责数据存储和操作。数据节点可以动态的扩容。架构图如下：



SequoiaDB作为“数据湖”可以连接两个世界

JSON文档的存储方式，适用于所有的结构化、非结构化和半结构化的数据。存储上，能将数据进行统一的存储。而SequoiaDB 对SQL还有多个大数据架构的完美支持，能够成为连接两种不同的数据操作方式的桥梁。



SequoiaDB：

- 结构化＋非结构化
- 独立部署,分布式部署
- 垂直扩展,水平扩展
- SQL + MapReduce
- 定量查询 + 定性预测
- 强关系 + 弱关系
- 商业软件 + 开源框架

高性能硬件平台加速大数据应用

PowerLinux高性能服务器

PowerLinux高性能服务器

- PowerLinux也称为Linux on Power，意即基于power架构的Linux服务器
- Power芯片基于RISC指令集架构（精简指令集），RISC服务器在硬件架构设计上与X86服务器有很大的差别，使用了非常多的冗余技术和高可用技术，因此可靠性较高，制作工艺也较难
- 最新一代power芯片power8针对大数据scaleout有着深度的优化，支持Redhat，Ubuntu，Suse等主流Linux操作系统

PowerLinux海量计算资源之SMT技术

- CPU并发多线程技术SMT（ Simultaneous multithreading ），是IBM 从Power5芯片开始实现的新技术，它支持一个CPU内核同时处理多条指令，因此可以最高达到单一CPU几倍的处理速度。 任何单个应用程序都不能完全使该处理器达到满负荷。当一个线程遇到较长等待时间事件时，同步多线程还允许另一线程中的指令使用所有执行单元。例如，当一个线程发生高速缓存不命中，另一个线程可以继续执行。

- Power8芯片在smt技术上做到并行8线程（smt=8），单个物理socket支持12core的前提下，也就意味着最高单颗cpu可以达到96threads，其海量的计算资源是大数据性能提升源源不断的动力。

Power8智能多线程（smt）

SMT1: Largest unit of execution work

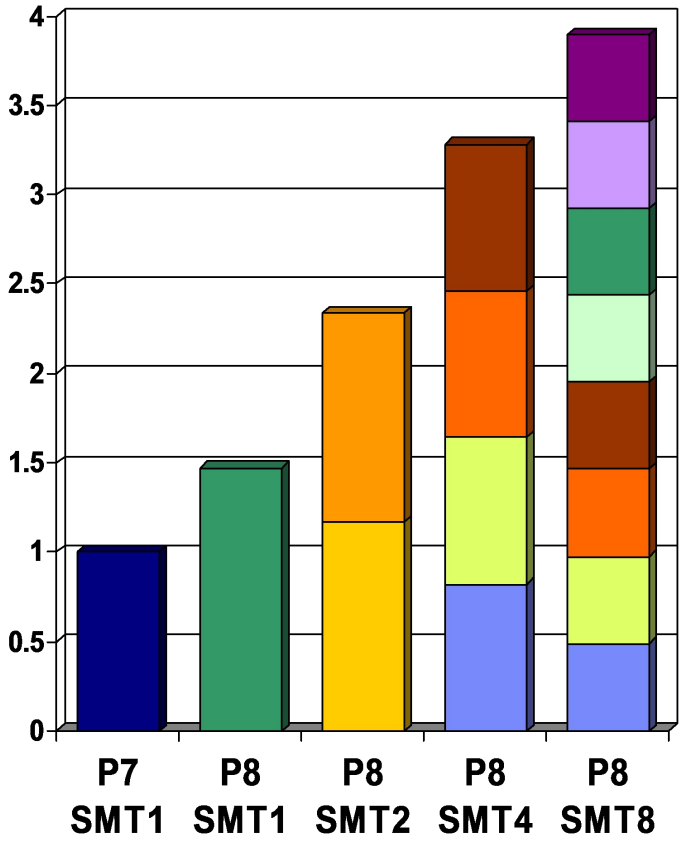
SMT2: Smaller unit of work, but provides greater amount of execution work per cycle

SMT4: Smaller unit of work, but provides greater amount of execution work per cycle

SMT8: Smallest unit of work, but provides the maximum amount of execution work per cycle

按需动态切换: SMT1 / SMT2 / SMT4 / SMT8

可以混合使用



小尾端支持Support Little Endian

Big/Little Endian

123

从左至右

一百二十三

Big Endian (BE)

Mainframe
RISC

从右至左

三百二十一

Little Endian (LE)

x86

Little Endian Support on Power8

Ubuntu 14.04.00/01

SUSE 12

Redhat7.1

用户收益

在应用迁移中, C/C++语言编写的Linux应用无需更改字节序的源代码，比如网络字节序与系统字节序间的转换。

Redhat、Ubuntu 和SUSE 的Little Endian 架构进一步减少了此类工作。

Power Virtualization Options

PowerVM

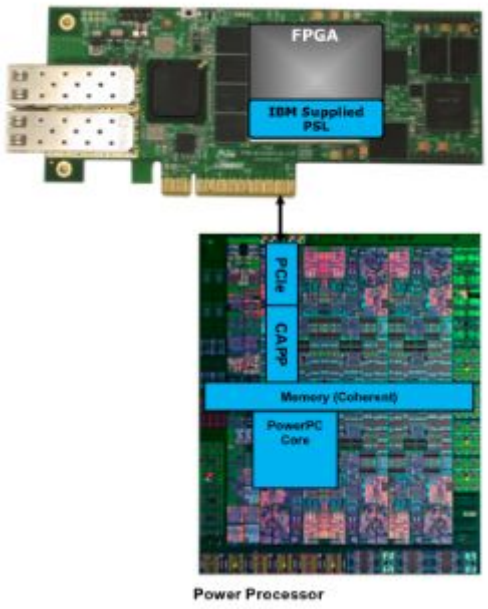
PowerVM: Provides virtualization of Processors, Memory, Storage, & Networking for AIX, IBM i, and Linux environments on Power Systems.

PowerKVM

PowerKVM: Open Source option for virtualization on Power Systems for Linux workloads.

For clients that have Linux centric admins. (RHEL 6.5 & SLES 11.3)

Power8支持CAPI协议和FPGA加速应用



DD Call

Copy or Pin Source Data

MMIO Notify Accelerator

Acceleration

Poll / Int Completion

Copy or Unpin Result Data

Ret. From DD Completion

Shared Mem. Notify Accelerator

Acceleration

Shared Memory Completion

40X fewer instructions

CAPI协议和FPGA加速

- Power8 提供了一组新的接口，称为CAPI(Coherent Accelerator Processor Interface)。通过CAPI接口，一些图形处理器（Nvida）或者FPGA等外部组件可以与Power8处理器直接通讯。CAPI接口是使用PCI-E3.0 通道提供。应用系统可以使用CAPI接入的这些图形处理器，FPGA等外部组件实现一些关键的功能。由于CAPI接入的外部硬件可以共享处理器的内存地址，所以减少了操作过程中由操作系统和不同层设备之间产生的堆栈路径长度，所以大幅提升运行效率和性能。实现了硬件加速功能。

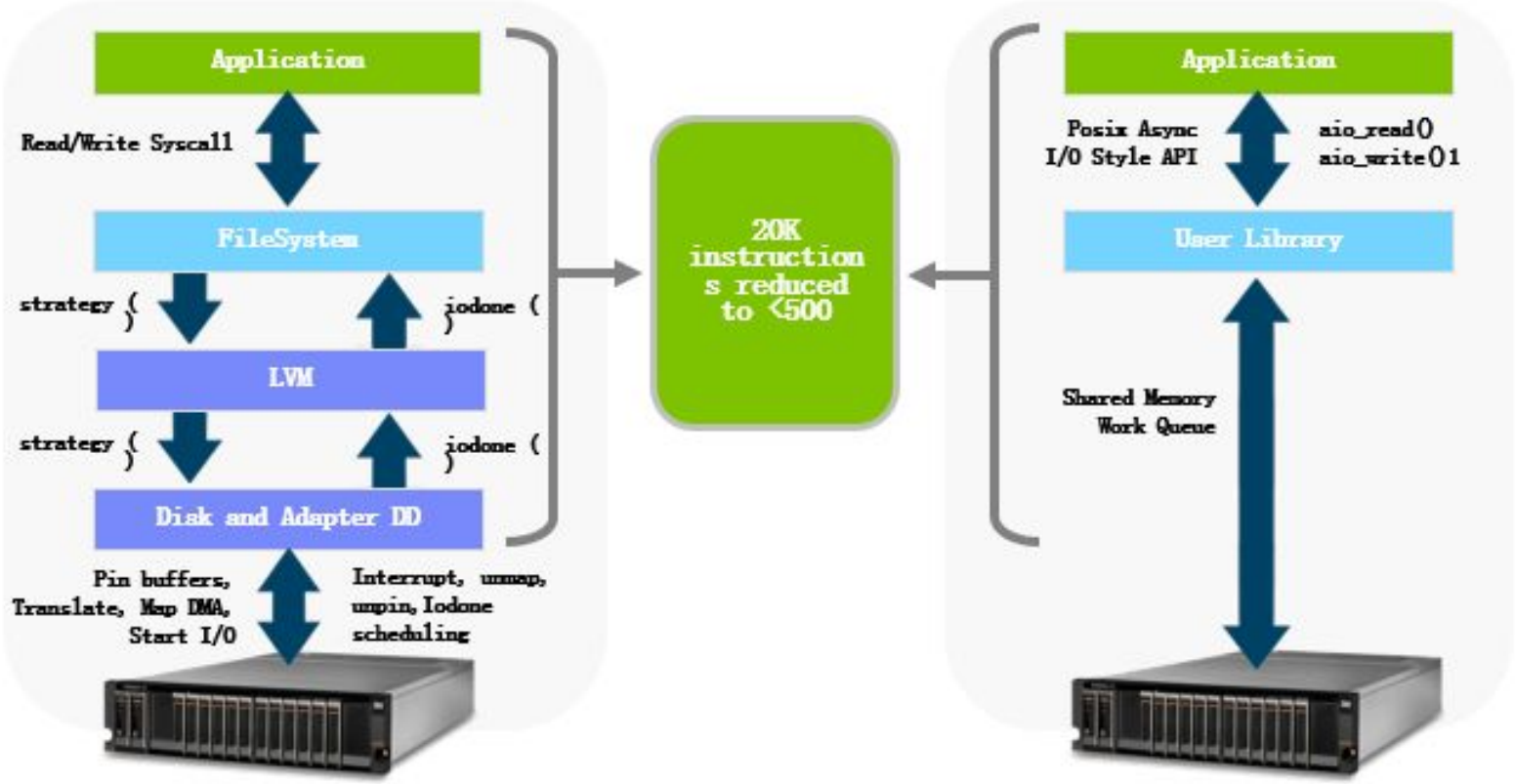
CAPI协议和FPGA加速

- 面对现在大数据分析、模式匹配、热点识别等，要求非常大的计算量，传统CPU出现瓶颈。普通的CPU其实也能计算，但是计算速度太慢，比如，只有64位宽来处理1GB数据，那便需要循环拆分N次才能算完，普通的CPU无法迅速处理这么大的运算量。

- 有人想到硬件加速，做法就是把某个专业计算在电路层面展开，展开成更宽的位宽，更多的并行计算单元，去除一些不必要的缓存优化和流水线优化等，其实这就是专用运算芯片所做的，FPGA(Field-Programmable Gate Array，即现场可编程门阵列)是现场可编程的专用芯片上述就是所谓的硬加速。

Power 8 CAPI Attached Flash Optimization solution

- Attach TMS Flash to POWER8 via CAPI coherent Attach
- Issues Read/Write Commands from applications to eliminate 97% of code pathlength
- Saves 20-30 cores per 1M IOPs



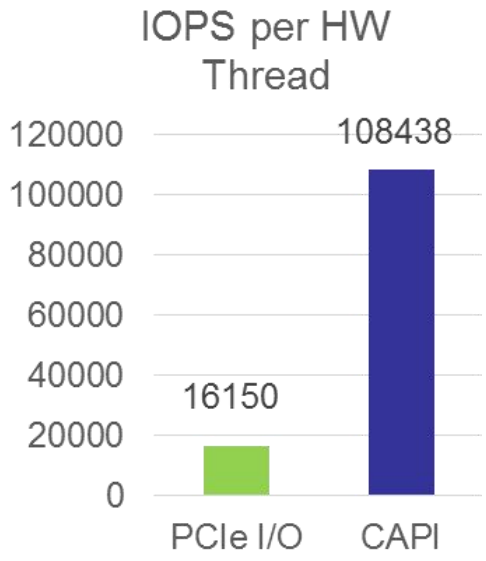
Data Engine for NoSQL

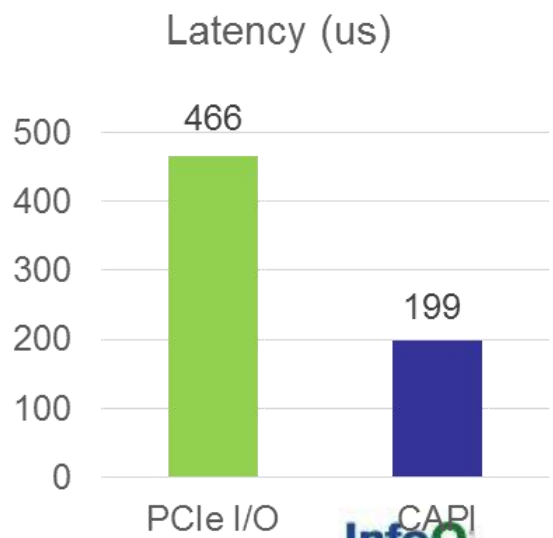
IBM基于支持CAPI+全闪存阵列而推出NoSQL数据加速引擎，配套全闪存阵列可以通过CAPI加速卡直接访问应用内存空间，大大降低了数据传输的延迟，非常有利于单笔数据访问量少，但IO密集的键值存储（KVS，Key-Value Store）平台

redis

CAPI

Flash System



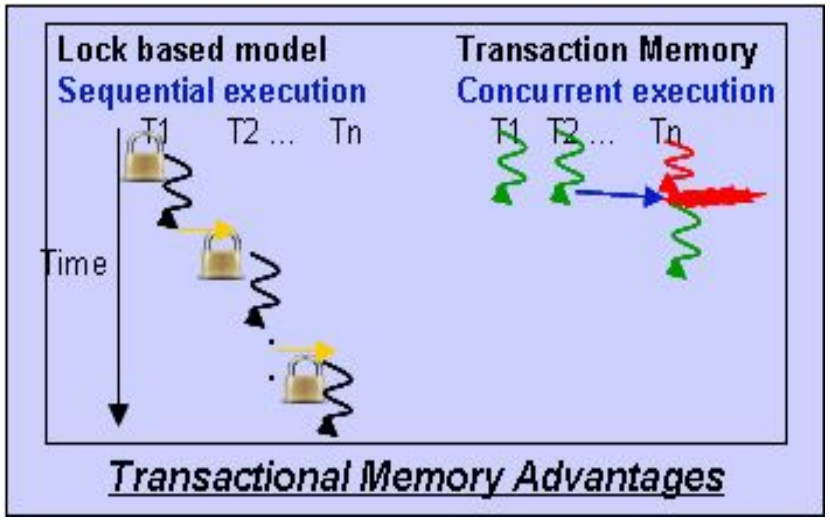


PowerLinux特色之事物内存

- 事务内存，是一种采取类似数据库事务技术实现的共享内存同步机制。事务内存的访问机制，允许进程/线程在访问临界区内存时，实现类似事务的原子特性——指令中的动作要么全部执行，要么一个都不执行。进程/线程使用两个新的CPU指令 tbegin和tend来标记一块即将访问的内存区域。
- Power8从硬件层面实现了对事务内存的支持。通过此技术的采用，避了使用锁时出现的一些诸如死锁等问题并极大的提高了并行度，有效地提升了应用系统的性能。

PowerLinux特色之事物内存

Transaction Memory



- POWER8 支持
  - 指令对事务(Transaction)的开始和结束进行标记
    - 硬件使用推测预判技术自动执行
  - 对于误判预测的恢复由硬件自动同时在内存和寄存器中执行
  - “扁平化” 嵌套
    - 硬件将其视为单一大型事务, 自动跟踪嵌套

事物内存与普通锁性能对比

Start 16 readers, and 4 writers, each will execute 1,000,000 transactions simultaneously.

1. Read-based read/write lock, implemented based on semaphore.

```
#./semrwlocktest 1000000 16 4 > sem.txt
real    1m2.20s                  32.7x Process Time
user    0m10.24s
sys     3m58.60s
```

2. Read-based read/write lock, implemented based on pthread read/write lock.

```
#./osrwlocktest 1000000 16 4 > os.txt
real    0m37.80s                 19.9x  Process Time
user    1m41.71s
sys     0m10.82s
```

3. Transactional memory only. when HTM failed, sleep an interval & retry. the interval is caculated using exponential backoff algorithm.

```
#./puretm 1000000 16 4 > puretm.txt
real    0m1.90s
user    0m5.98s
sys     0m0.00s
```

大数据技术架构的优化

Spark on Power优化

Spark On PowerLinux之优化方案

- 重点关注：
- 1. spark.local.dir多路径配置，以及ssd、HDD、memdisk的选择
- 2. spark.executor.memory大小配置与cores、smt的配比
- 3.配置多个executor来充分利用机器强悍的性能
- 4. memoryFraction 数值的选择，决定了用于缓存中间数据的内存
- 5. JVM和GC策略的的选择，G1 Garbage Collector 或者 Parallel GC

spark.local.dir多路径配置，以及ssd、HDD、memdisk的选择

- ✓ 在配置文件中配置多个路径将中间结果（RDD Cache，Shuffle等数据）散落到多块磁盘上，增加IO带宽。注意多路径通过逗号隔开。
- ✓ 为了提升性能加速读写，可以采用HDD+SSD混用的方式，尽量多的将路径配置到SSD上来提升IO能力。
- ✓ 在有充足的内存空间的情况下，可以配置memdisk，来获取极限的速度，但需要注意内存空间毕竟不必磁盘，防止中间数据过大将内存沾满，其次，此种方法也会一定程度耗用内存带宽，会影响一部分内存速度。

spark.executor.memory大小配置与cores、smt的配比

- ✓ Executor 内存的大小，和性能本身当然并没有直接的关系，但是几乎所有运行时性能相关的内容都或多或少间接和内存大小相关。这个参数最终会被设置到Executor的JVM的heap尺寸上，对应的就是Xmx和Xms的值

- ✓ 理论上Executor 内存当然是多多益善，但是实际受机器配置，以及运行环境，资源共享，JVM GC效率（每个executor分配过大的heap size会增大GC负载）等因素的影响，还是需要为它设置一个合理的大小。

- ✓ Smt智能多线程技术可以根据现场情况灵活选择，一般采用smt4或8

配置多个executor来充分利用机器的性能

- ✓ 通常情况下，单台服务器启动1个executor，并配置好相应的memory、cores足以充分调用机器的资源。
- ✓ PowerLinux可能打破了常规认识。在实际使用的情况中，Power上需要启动多个executor（最高达到7个！）才能够完全调动系统资源。
- ✓ 并不是executor个数越多越好，过多的executors会带来task之间的通信开销加大，反而会降低性能。

memoryFraction 数值的选择，决定了用于缓存中间数据的内存

- ✓ spark.executor.memory决定了每个Executor可用内存的大小，而spark.storage.memoryFraction则决定了在这部分内存中有多少可以用于Memory Store管理RDD Cache数据，剩下的内存用来保证任务运行时各种其它内存空间的需要。
- ✓ 集群task并行度：SPARK_ EXECUTOR_INSTANCES* SPARK_EXECUTOR_CORES；每个task运行能使用到的内存：(ExecutorMem *(1 - MemFraction)) / SPARK_EXECUTOR_CORES。
- ✓ 优化缓存大小：默认情况Spark采用运行内存（spark.executor.memory）的60%来进行RDD缓存。这表明在任务执行期间，有40%的内存可以用来进行对象创建。如果任务运行速度变慢且JVM频繁进行内存回收，或者内存空间不足，那么降低缓存大小设置可以减少内存消耗，可以降低spark.storage.memoryFraction的大小。

JVM和GC策略的的选择，G1 Garbage Collector 或者 Parallel GC

作为一种以内存为中心的计算引擎， Spark甚至使用超过100G或更大内存作为Heap Size，这在传统的应用上是不可想象的。因此，JVM以及GC策略的选择和调优往往直接影响到Spark程序的运行。

☐ Open JDK

- 兼容性好，但是没有针对Power硬件优化

☐ IBM JDK

- 性能表现良好，但仅限于Power平台表现较好

不同JVM的性能对比

Garbage Collector	Running Time
Parallel GC	6.5 min
CMS GC	9 min
G1 GC	7.6 min (Before Tuning) 4.3 min (After Tuning)

* 使用**Oracle JDK**运行一个基于Spark Bagel的多次迭代应用测试结果 [1]

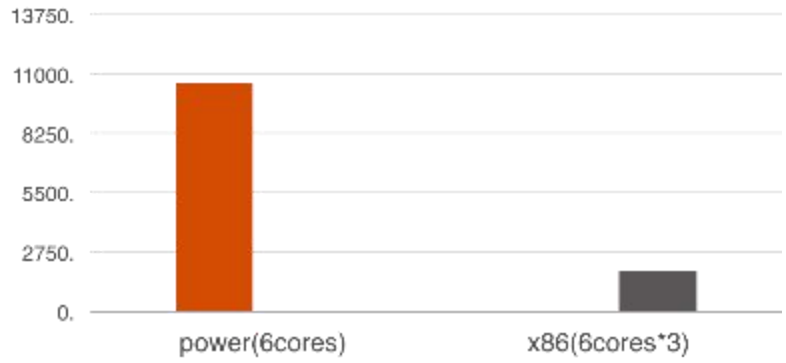
GC Policy	Running Time
gencon	4.7 min
subpool	3.0 min (After Tuning)

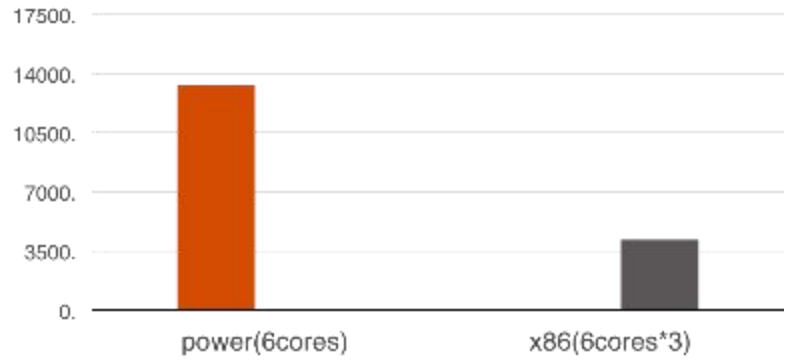
* 使用**IBM JDK**在Power S822L上运行某客户的场景测试结果

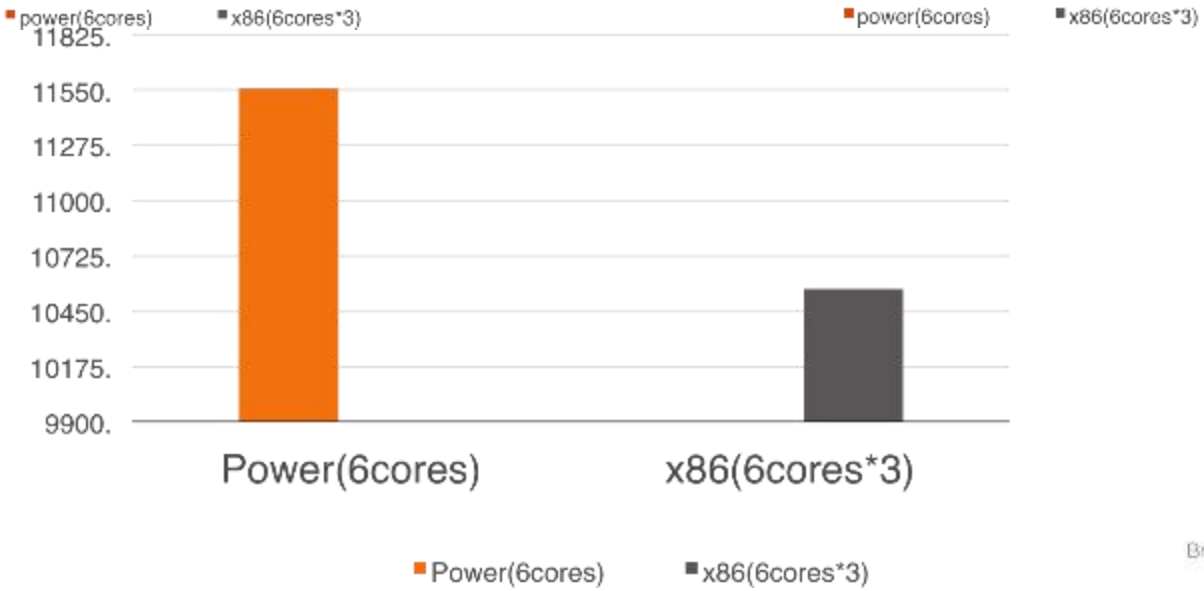
数据来源:

[1]. https://databricks.com/blog/2015/05/28/tuning-java-garbage-collection-for-spark-applications.html

SequoiaDB 在 Power Linux下的优异性能





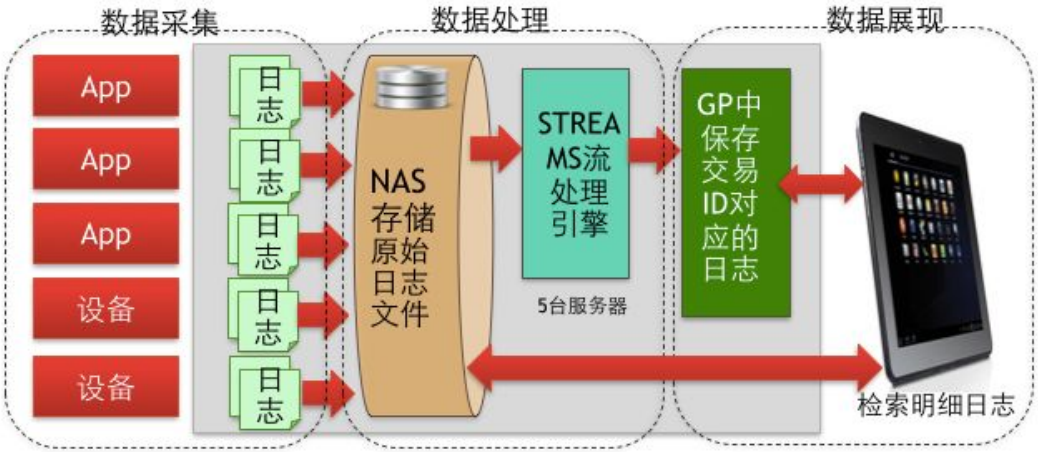


数据分析平台案例

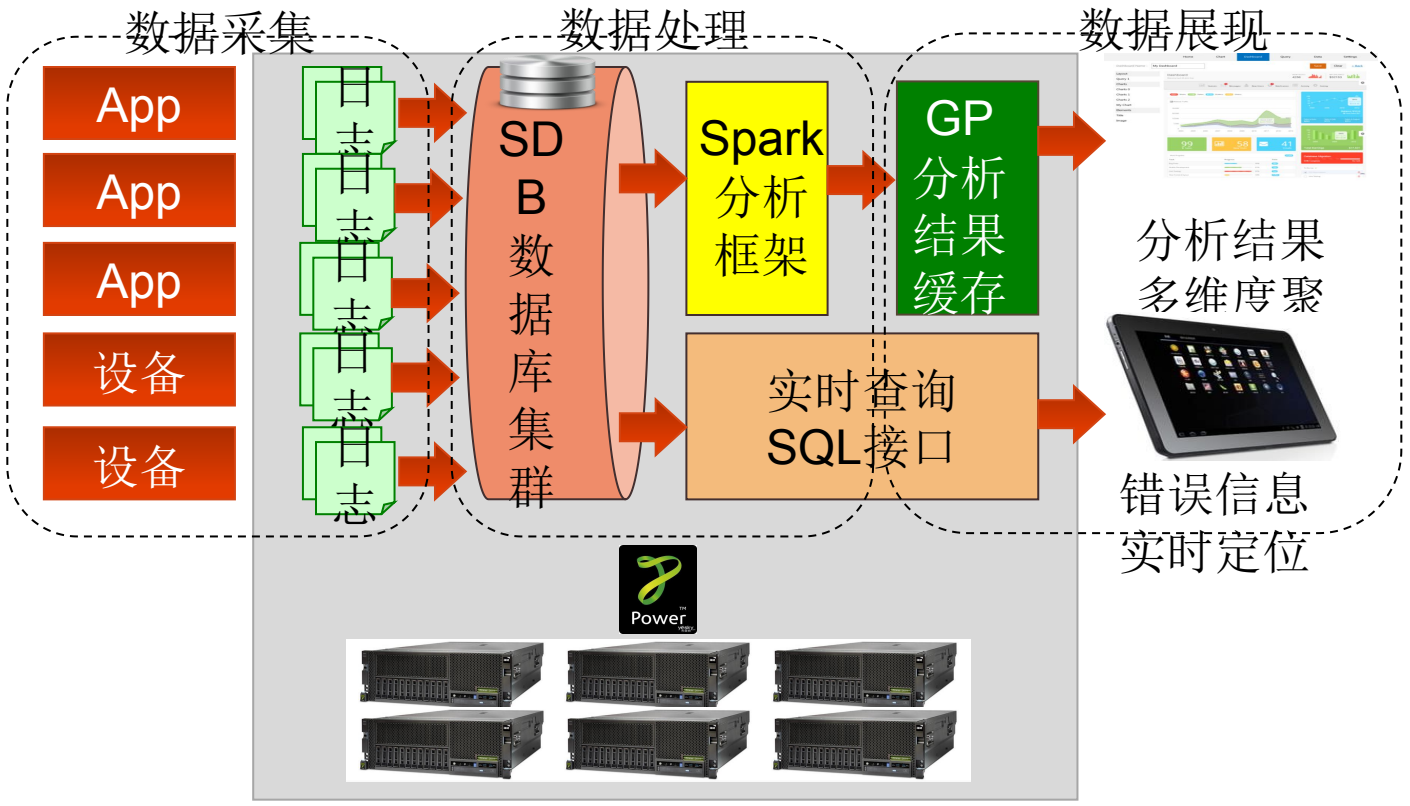
PowerLinux+SequoiaDB+Spark

Power+SequoiaDB+Spark 大数据海量日志平台

某电信运营商，拥有海量的日志系统，数据在峰值大约为每小时300GB，而计算生成的数据量约为500万条。原有的NAS存储日志的架构无法处理每日TB级别的日志增长需求。同时，硬件设备也极大地制约了其效率。



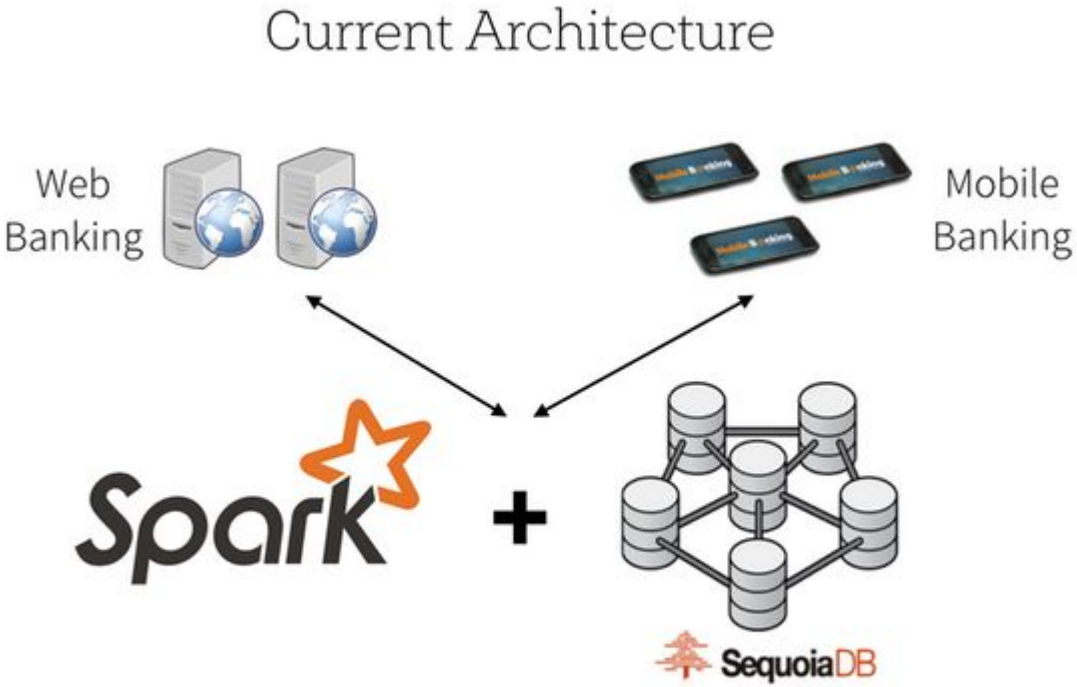
在构建新系统时，使用SequoiaDB作为日志文件的存储，替换原有的NAS+HDFS的日志存储架构，Spark分析架构替换原有的Streams流处理引擎，服务器选用了Power 8 来实现。



Power+SequoiaDB交易历史存储系统

在过去的数十年中，大部分银行都一直使用大型机作为其核心的银行业务系统支持。大型机的技术局限性使得超过一年之前的历史交易数据就需要从大型机上移出而转存在其他的磁盘（磁带）中进行保存。
然而，如今，因为移动互联网和网上银行的兴起，银行用户对于服务的要求大大的超过了从前，这也带来了更多的需求。为了让客户更好的体验银行的服务，让产品服务更有竞争力，各大银行也开始推出让客户能快速的查询历史记录（包括1年以前的历史记录）等多项改进的服务。

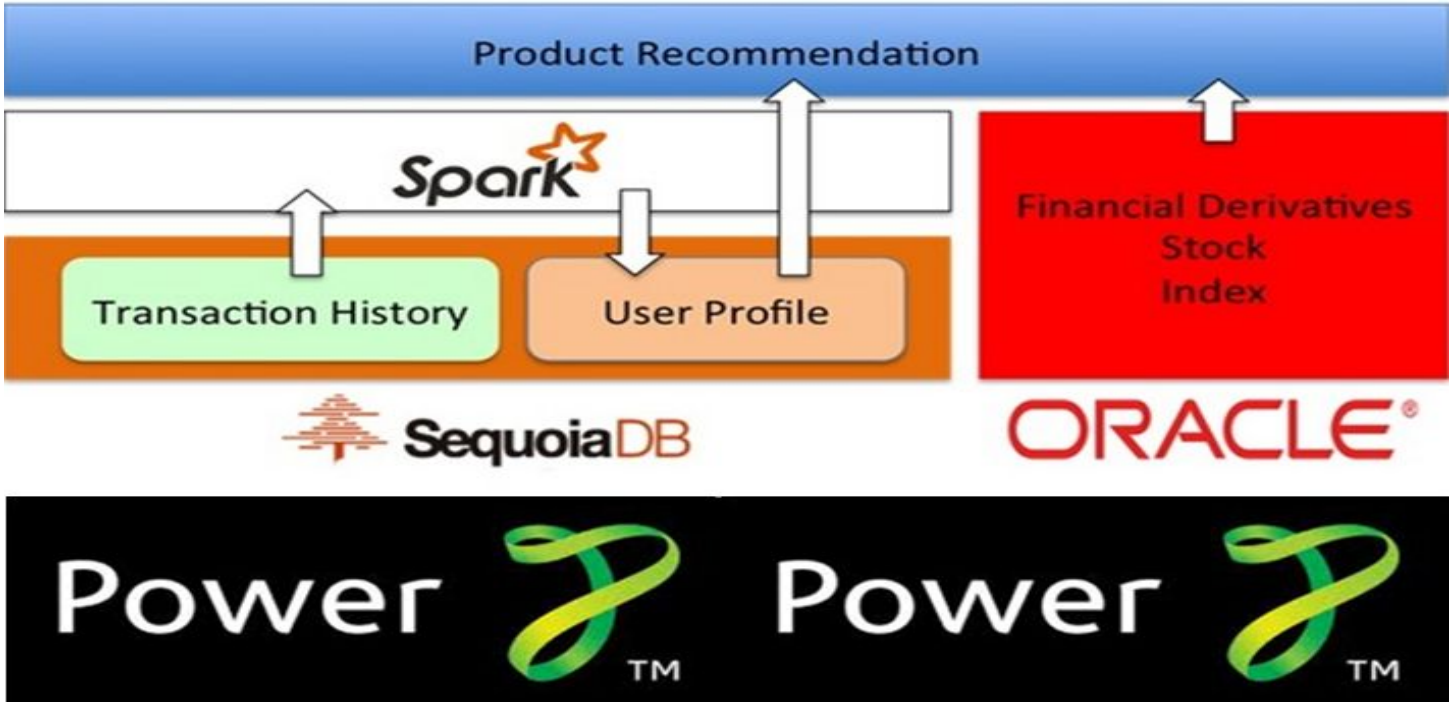
通过使用SequoiaDB和PowerLinux服务器，该银行在数据库的50个物理节点，使用近1PB的空间，存储了所有用户长达15年的历史数据。这一新系统让用户可以轻松的获取其所有的交易历史，无论展现在移动客户端或者网页端。



产品精准推荐系统

在之前的案例中，我们已经将所有的用户历史交易数据都存储在了数据库当中，基于这些历史交易信息，我们也可以通过对这些数据的分析，对每个用户的交易行为进行预测，对用户进行分类和建模，最终根据分析的结果向每个用户推荐最适合的理财产品。

当用户模型系统通过分析所有的历史数据和日志，计算出需要推荐的产品时，这些用户特征也会作为这个用户的一个标签写入这个用户的信息中。这些新加入的用户标签，可以帮助前台的员工和产品推荐系统快速的分辨出每个顾客的兴趣和消费倾向。部署了这套系统后，该行的金融产品的推荐成功率提升了10倍以上。



THANKS