## 数据库性能需求分析及评估模型

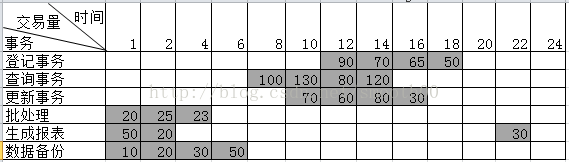
       数据库作为应用系统当中最重要的一块，也是性能测试非常关注的一块，根据我自己的项目经验，和以往对应用系统的性能需求分析和测试策略制定过程，总结一下如何开展数据库系统的性能需求分析，以及制定数据库能力评估模型。

### 一、数据库性能需求制定

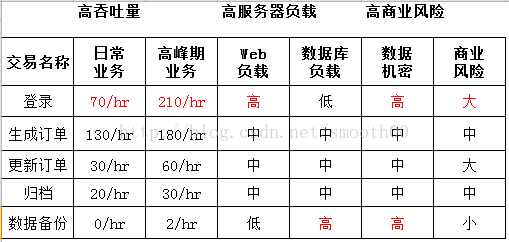
#### 1、需求信息收集-任务/交易分布

（1）收集有哪些主要交易任务（与业务系统需求一致）

（2）在一天的某些特定时刻系统都有哪些主要操作，以及操作量



#### 2、需求信息收集-交易混合图



      需要关注的信息有：

Ø   高峰期有哪些操作？

Ø   中间件操作有多少？数据库操作有多少？

Ø   如果任务失败，那么商业风险有多少？

Ø   有没有涉及保密系数高的数据？

      测试选型标准：高吞吐量、高I/O、高商业风险

### 二、数据库能力需求

1、高吞吐量

      满足高并发下的大数据量交互需求，满足数据备份或ETL过程的大数据量迁移。具体需求信息获取参照以上数据库应用需求。

2、负载均衡

      满足高并发下数据库的负载均衡能力，需求分析需要收集数据库的部署架构、负载均衡策略等数据信息。

3、读写分离

      获取需求的要点是明确哪些是写节点，哪些是读节点，并且切换的策略什么，数据同步的策略是什么。

4、分区分片（分库分表）

      获取需求的要点是把握数据的垂直切换和水平分库概念。明确需要对哪些数据块进行切分，分别分散到哪几台数据库主机上；需要对哪些大表进行数据水平切分，并且分布到哪些DB或table中。通过需求分析，做出数据切分的合理性判断，以及做出系统可测性的判断。

5、高并发

      根据以上的数据库应用需求，进一步制定数据库的高并发需求，估算出单台数据库的API接口压力和需要满足的并发能力。

6、高可用性

      高可用性可能也综合涉及到数据的多项能力，主要应用的是集群技术，HA容错及互备技术，体现的是无故障运行。获取需求的要点是明确高可用性技术架构，了解HA采用的工作方式，以及掌握故障切换方法和数据一致性验证需求。

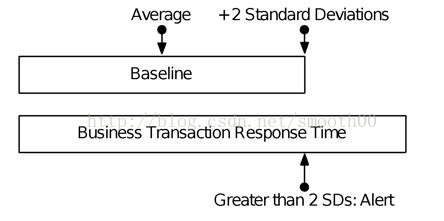
### 三、数据库评估模型

#### （一）关键业务时间指标

       在我们的印象中，应用的关键业务能够提供真正的用户行为洞察——他们捕捉实时性能数据，展现真实用户在交互时的用户体验。衡量一个关键业务的性能包括捕捉交易的整体响应时间以及测量其不同层面的响应时间。这些时间都可以满足你业务需要的基准做比较。

      如果你只想测量应用的某个方面，关键业务流程是最佳选择。虽然容器指标可以提供丰富的信息，可以帮助你确定何时自动缩放您的环境，但业务流程交易还是决定着你的应用性能最终效果。不管你作为什么规模公司的程序“猿”，都应该首先关心你的用户是否可以完成他们的关键业务流程。

       一旦你定义了整个关键业务，性能好坏就是衡量整个应用生态系统的最好标准。我们需要设定低于平均关键业务响应时间的交易为异常行为，这样就能更好的观察应用的性能了，如下图所示。



      那么问题来了，怎么设定关键业务的标准呢？

      这里提供一个简单的方法供大家参考：假设关键业务在周三13：00~14:00是一个常见的高峰，那么选择这个时段平均响应时间作为标准，等到下周三的同一个时段，再将这周的这个时段的所有关键业务平均响应时间与前一周相比得到一个平均值，如此循环。通过这个机制，应用可以随时间而发展，而原始的关键业务数据也变得更有意义。关键业务的监测是用户体验中最具反思性的测量方法，因此它们是能捕捉到的最重要的指标之一。

#### （二）SQL性能指标

      查询的性能主要体现为SQL查询缓慢和数据返回时间过长。那么我们要怎么解决它呢？下面是测试需要重定关注的：

      1、数据的查询方式对传输效率的影响，比如选用了更多的数据：查询返回的列太多的话会导致在选择行和检索数据时造成缓慢（如使用了SELECT\*，没有列出所需的列）。此外，在结果中列出所需的列，也能减少数据传输，有利于性能的提升。

      2、重点关注索引的应用，对于只是访问表中的几个字段，并且字段内容较少，可以为这几个字段单独建立一个组合索引，这样就可以直接只通过访问索引得到数据，一般索引占用的磁盘空间比表小很多，所以这种方式可以大大减少磁盘IO开销。

      3、关注慢SQL执行计划及优化：SQL执行计划是关系型数据库最核心的技术之一，它表示SQL执行时的数据访问算法。当业务需求越来越复杂，表数据量越来越大，SQL也需要支持非常复杂的业务逻辑，但SQL的性能还需要提高，因此，优秀的关系型数据库除了需要支持复杂的SQL语法及更多函数外，还需要有一套优秀的算法库来提高SQL性能。

      4、关注批处理对性能影响：读取大量的数据或生产复杂的分析报告时通常都是在批量操作。这些操作是资源密集型，可能会影响在线用户的性能。想要解决这个问题需要将这些操作在低负载下进行，如在夜间；或使用单独的数据库来处理和分析报告。

#### （三）锁及粒度

      数据库一般都允许多用户的存在，多个用户同时活动必然会导致冲突，然而正常工作中这种情况又无法避免，所以测试需要关注的是锁的应用与性能的平衡关系：

      1、页/行锁定：当一个用户试图读取另一个用户正在修改的数据，或修改另一个用户正在读取的数据时，又或者尝试修改另一个事务正在尝试修改的数据时，就会出现并发问题。这时候资源就会被锁定。

      可以锁定的资源在粒度（granularity）上差异很大。从细（行）到粗（数据库）。细粒度锁允许更大的数据库并发，因为用户能对某些未锁定的行执行查询。然而，每个由数据库系统产生的锁都需要内存，所以数以千计独立的行级别的锁也会影响数据库的性能。粗粒度的锁降低了并发性，同时消耗的资源也较少。

      2、死锁：死锁是数据库性能的重量级杀手之一，然而死锁却是不同事务之间抢占数据资源造成的。死锁耗时耗资源，然而在大型数据库中，高并发带来的死锁是不可避免的，所以我们只能让其变的更少。

①按照同一顺序访问数据库资源

②保持事务的简短，尽量不要让一个事务处理过于复杂的读写操作。事务过于复杂，占用资源会增多，处理时间增长，容易与其它事务冲突，提升死锁概率。

③尽量不要在事务中要求用户响应，比如修改新增数据之后再完成整个事务的提交，这样延长事务占用资源的时间，也会提升死锁概率。

④尽量减少数据库的并发量（通过优化架构实现）。

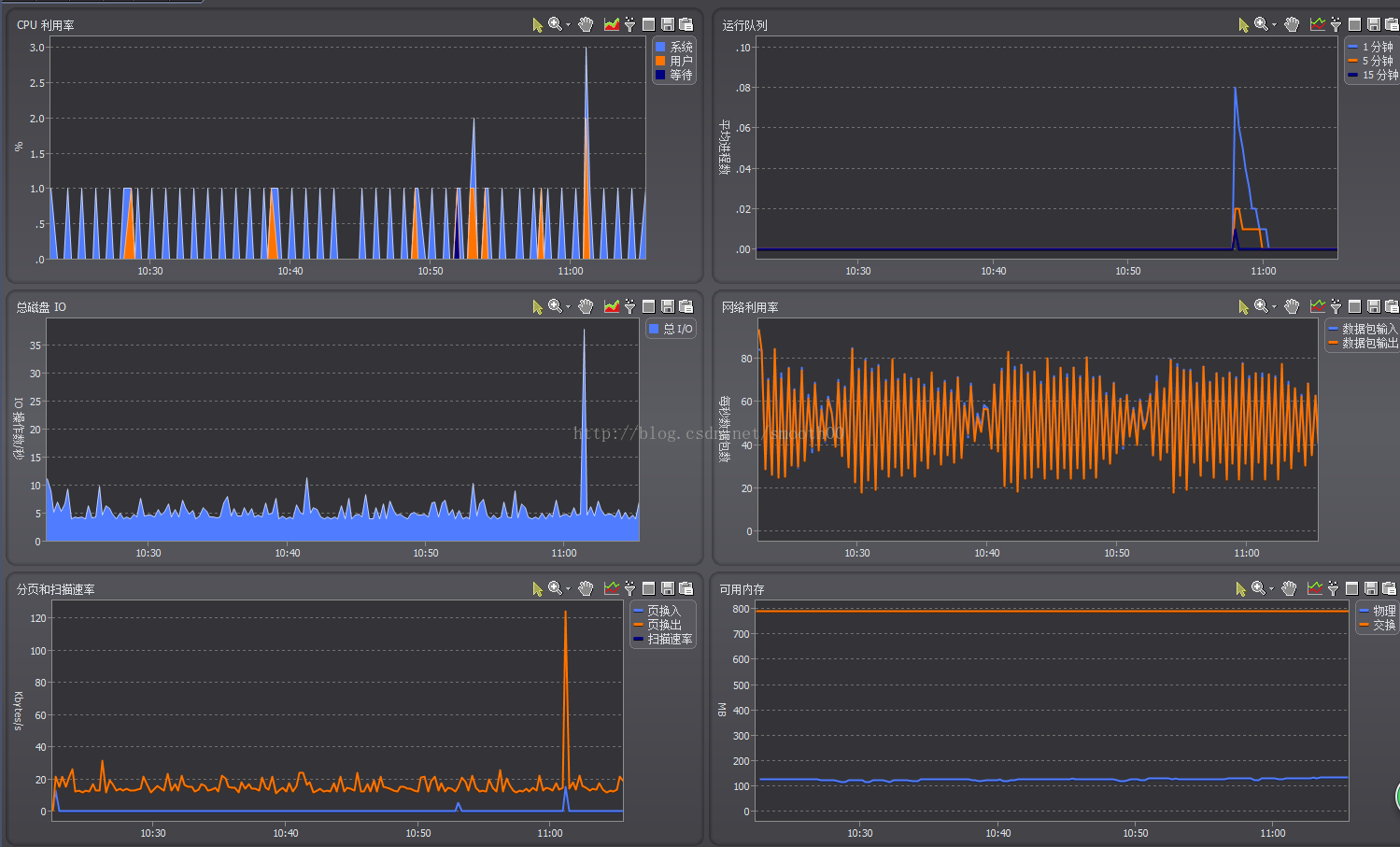
⑤尽可能使用分区表，分区视图，把数据放置在不同的磁盘和文件组中，分散访问保存在不同分区的数据，减少因为表中放置锁而造成的其它事务长时间等待。

⑥避免占用时间很长并且关系表复杂的数据操作。

⑦使用较低的隔离级别，使用较低的隔离级别比使用较高的隔离级别持有共享锁的时间更短。这样就减少了锁争用。

#### （四）硬件资源指标

      然而并不是所有的数据库性能问题都是来自数据库本身，我们日常工作中最常见的另一个情景就是数据库的硬件有若干问题，这里我们简单的介绍一下可能会出现的情况，毕竟市面上有已经有很多工具可以监测这些问题了。

1、没有足够的CPU或CPU速度太慢：更多的CPU可以分担服务器的负载，从而提高性能。

2、慢的磁盘没有足够的IOPS：磁盘性能可以描述为每秒输入/输出操作（IOPS），它表示每秒磁盘的吞吐量。

3、配置不正确的磁盘：数据库需要效果明显的磁盘访问，配置不正确的磁盘会造成相当大的性能影响。

4、没有足够的内存：受限或不好的物理内存影响数据库性能，可用的内存越多，性能越好。

#### （五）NoSQL

      NoSQL发展到今天，已经有了很大的吸引力，因为它处理大规模数据和高并发的能力非常显著。但是，NoSQL却很难测试，也不容易监控。

1、NoSQL特性：关系型SQL已经成熟，有行业标准接口，但是每一个NoSQL都是独一无二的存在，并且都不支持复杂的数据库模型。所以简洁、有效、速度是它的业务应用标准。

2、集群化和负载均衡：NoSQL数据库相比关系型数据库通常更多的是资源密集型。它们需要更多的内存和内存分配，集群化和分布式是评估要点；

3、扩展性要求：随着数据库的需求增加，硬件也必须扩展，可扩展性是一项评估要点。

4、高可用性要求：可以说NoSQL对稳定性的要求更为苛刻（因为它们有些是基于内存的数据库），高可用性是重要评估点。

5、监控要求：相对于已经成熟的关系SQL，NoSQL现在的监控可以说是比较困难的，国内也只有听云一家公司能够支持主流的Memcached, MongoDB, Redis等非关系型数据库服务（但是NoSQL监控部分要收费）；ApplicationsManager也支持对Memcached, MongoDB,Redis、HBase、Oracle NoSQL的监控（这些的监控指标还不够丰富，有待完善）。

#### （六）扩展架构模型

##### 1、数据切分和分布式

      数据切分可以是物理上的，对数据通过一系列的切分规则将数据分布到不同的DB服务器上，通过路由规则路由访问特定的数据库，这样一来每次访问面对的就不是单台服务器了，而是N台服务器，这样就可以降低单台机器的负载压力。数据切分也可以是数据库内的，对数据通过一系列的切分规则，将数据分布到一个数据库的不同表中。

（1）数据垂直切分

      数据的垂直切分，也可以称之为纵向切分。将数据库想象成为由很多个一大块一大块的“数据块”（表）组成，我们垂直的将这些“数据块”切开，然后将他们分散到多台数据库主机上面。这样的切分方法就是一个垂直（纵向）的数据切分。

（2）数据水平切分

      数据的垂直切分基本上可以简单的理解为按照表、按照模块来切分数据，而水平切分就不再是按照表或者是功能模块来切分了。一般来说，简单的水平切分主要是将某个访问极其平凡的表再按照某个字段的某种规则来分散到多个表之中，每个表中包含一部分数据。

除了垂直切分、水平切分，还有其他的切分或分片方式，如导出切分、混合切分。

（3）负载均衡和读写分离

      一般是通过负载均衡器，其职责就是定位到一台具体的DB服务器。具体的规则如下：负载均衡器会分析当前sql的读写特性，如果是写操作或者是要求实时性很强的操作的话，直接将查询负载分到Master，如果是读操作则通过负载均衡策略分配一个Slave。

      其中Master负责写操作的负载，也就是说一切写的操作都在Master上进行，而读的操作则分摊到Slave上进行。这样一来的可以大大提高读取的效率。在一般的互联网应用中，经过一些数据调查得出结论，读/写的比例大概在 10：1左右 ，也就是说大量的数据操作是集中在读的操作，这也就是为什么我们会有多个Slave的原因。

      但是为什么要分离读和写呢？熟悉DB的技术人员都知道，写操作涉及到锁的问题，不管是行锁还是表锁还是块锁，都是会降低系统执行的效率。我们这样的分离是把写操作集中在一个节点上，而读操作在其他的N个节点上进行，从另一个方面有效的提高了读的效率，保证了系统的高可用性。

（4）分布式存储

      分布式存储是将数据分散存储在多台独立的设备上。传统的网络存储系统采用集中的存储服务器存放所有数据，存储服务器成为系统性能的瓶颈，也是可靠性和安全性的焦点，不能满足大规模存储应用的需要。分布式网络存储系统采用可扩展的系统结构，利用多台存储服务器分担存储负荷，利用位置服务器定位存储信息，它不但提高了系统的可靠性、可用性和存取效率，还易于扩展。

      分布式存储利用的就是数据的切分，也叫数据分片，数据分片将达到以下三个目的：

Ø 分布均匀，即每台设备上的数据量要尽可能相近；

Ø 负载均衡，即每台设备上的请求量要尽可能相近；

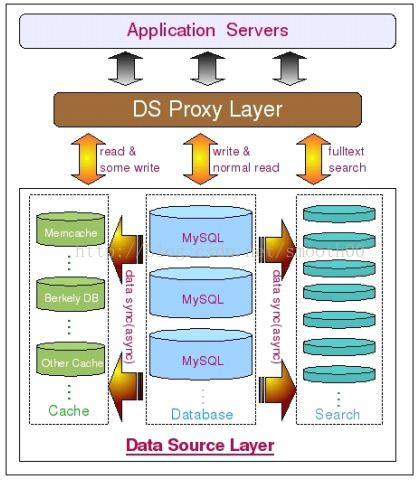
Ø 扩缩容时产生的数据迁移尽可能少。

      有了分布式存储，就会有分布式计算，这就是大数据的范畴了，在这里不多说。

##### 2、Cache与Search的利用

      （1）结合传统关系型数据库和NoSQL两种类型数据库的优缺点，对于Oracle、Mysql这些数据库，可以通过引入Cache（Redis、Memcached），减少数据库的访问，增加性能（主要是解决传统关系型数据库的IO性能瓶颈，Cache都是基于内存的，大大减少了磁盘读写）。特别说明一下，这里说的Cache不是指数据库底层对应的Cache缓存，数据库层次的缓存一般针对的是查询内容，而且粒度也太小，一般只有表中数据没有变更的时候才发挥作用。我们这里说的Cache，是指外部引入的数据库缓存。

      （2）通过引入Search（Lucene、Solr、ElasticSearch），利用搜索引擎高效的全文索引和分词算法，以及高效的数据检索实现，来解决数据库和传统的Cache软件完全无法解决的全文模糊搜索、分类统计查询等功能。



      通过以上的数据库性能评估模型分析，我们就能把握数据库系统将要有的性能能力，并分析具体的测试范围和指标评估范围，以决定后期需要采用的测试方法、策略和工具。毕竟性能不是靠测试和调优出来的，是靠设计出来的，如果我们不了解数据库的能力模型和相应的架构要求，我们将很难深入开展相应的性能测试和性能调优工作。