**Chapter 8. Performance**

理解：性能概念。

**Performance：**性能是关于时间和软件系统满足时序要求的能力。当事件发生时 （中断，消息，来自用户或其他系统的请求，或标记时间流逝的时钟事件）系统必须及时响应它们。表征可能发生（当它们可以发生）事件和系统，或对这些事件元素的基于时间的反应是在讨论performance的本质。

**Chapter Summary:** 性能是关于系统资源的管理，面对特定类型的需求，以实现可接受的计时行为。可以根据交互式和嵌入式实时系统的吞吐量和延迟来衡量性能。通过减少需求或更合理地管理资源可以提高绩效。

了解：性能公式。性能一般场景。

一般场景：

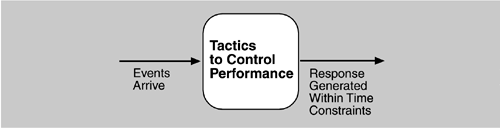
|  |  |
| --- | --- |
| **Portion of Scenario** | **Possible Values** |
| Source | 系统的内部或外部 |
| Stimulus | 周期性、偶发性或随机性事件的到来 |
| Artifact | 系统或系统中的一个或多个组件。 |
| Environment | 运行模式：正常，紧急，峰值负载，过载。 |
| Response | 处理事件，改变服务水平 |
| Response Measure | 延迟，截止日期，吞吐量，抖动，未命中率 |

Example：用户在正常操作下启动事务。 系统以平均延迟2秒处理事务。

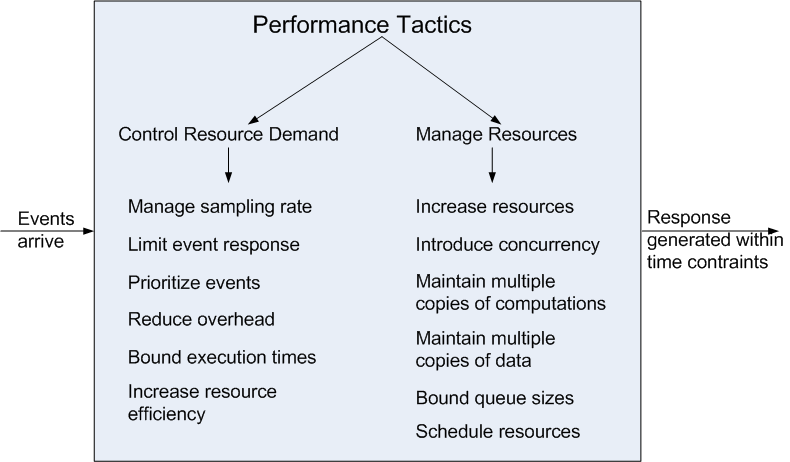
掌握：性能战术。性能设计清单。

Goal of performance tactics：

在一些基于时间的约束内生成对到达系统的事件的响应。



Performance tactics:



1. **Control resource demand**
2. **Manage sampling rate(**采样率**):** 如果可以降低捕获数据流的采样频率，则可以降低需求，通常会丢失一些保真度。
3. **Limit event response:** 只有在设置的最大速率下处理事件，才能确保在实际处理事件时进行更可预测的处理。
4. **Prioritize events：**如果并非所有事件都同等重要，您可以强制执行优先级方案，根据事件的重要性对事件进行排序。
5. **Reduce overhead** (开销)：使用中介增加了处理事件流所消耗的资源; 删除它们可以改善延迟。
6. **Bound Execution Times**：限制用于响应事件的执行时间。
7. **Increase Resource Efficiency**：改进关键区域中使用的算法将减少延迟。
8. **Manage resources**
9. **Increase resources:** 更快的处理器，额外的处理器，额外的内存和更快的网络都有可能减少延迟。
10. **Increase Concurrency(并发性)：**如果可以并行处理请求，则可以减少阻塞时间。
11. **Maintain Multiple Copies of Computations (维护多个计算副本)：**副本的目的是减少在单个服务器上进行所有计算时可能发生的争用。
12. **Maintain Multiple Copies of Data:** 使用不同的访问速度保留数据的副本（可能是另一个的子集）。
13. **Bound Queue Sizes:** 控制排队到达的最大数量，从而控制用于处理到达的资源。
14. **Schedule Resources：**当存在资源争用时，必须安排资源。

Design for performance:

1. 确定系统的职责，包括繁重的负载，具有时间关键的响应要求，大量使用，或影响系统中发生重负载或时间紧急事件的部分。确定:

处理每个要求并确定它们是否可能造成瓶颈 ； 适当识别和处理请求的其他责任(包括: 跨过程或处理器边界的责任, 管理线程的责任, 调度共享资源或管理与性能相关的队列，缓冲区和缓存的职责)

1. 确定必须与每个系统协调的系统要素，选择通信和协调机制

支持任何引入的并发，事件优先级或调度策略 ；确保可以提供所需的性能响应；具有通信机制的适当属性，例如，有状态，无状态，同步，异步，保证传递，吞吐量或延迟。

1. 确定数据模型中将会负载多重，具有时间关键响应要求，大量使用或影响系统发生重负载或时间紧急事件的部分。 对于那些数据抽象，确定：

是否保留关键数据的多个副本将有利于提高性能； 分区数据有利于提高性能 ；是否可以减少处理需求/添加资源以减少枚举数据抽象的创建，初始化，持久性，操作，转换或破坏的瓶颈。

1. 确定系统中哪些资源对性能至关重要。对于这些资源，确保在正常和过载的系统操作下对它们进行监视和管理。

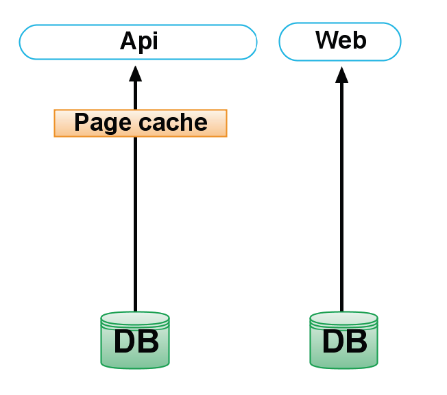
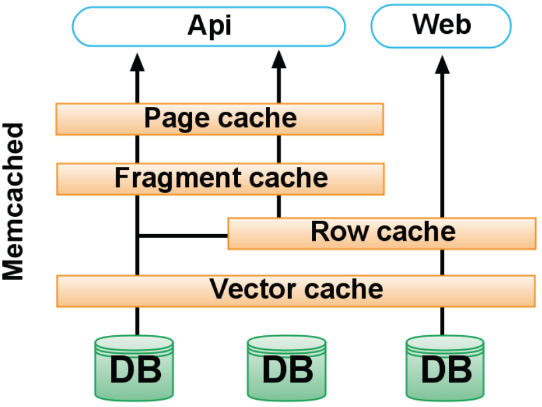
需要了解和管理的系统元素; 时间和其他性能关键资源；流程/线程模型；优先考虑资源和获取资源；调度和锁定策略；按需部署额外资源以满足增加的负载

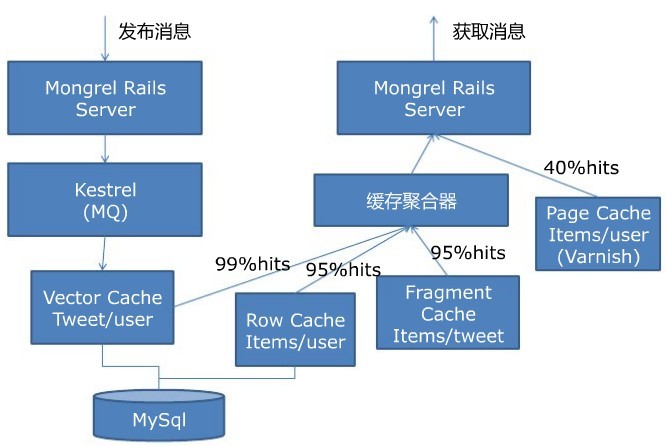
Case study（看看就好系列）：

* Twitter



Evolving architecture, cache ：

(数据有给)

* Facebook's Challenge

