Twitter storm 性能测试报告

测试目的

测试 twitter storm 的运行性能以及数据处理的延迟。

系统配置

CPU 个数	16
CPU 主频	2.26GHz
内存	16G
OS	Linux 2.6.32-xxx.x86_64
网络	千兆以太网
Storm 版本号	0.6.1

测试方法

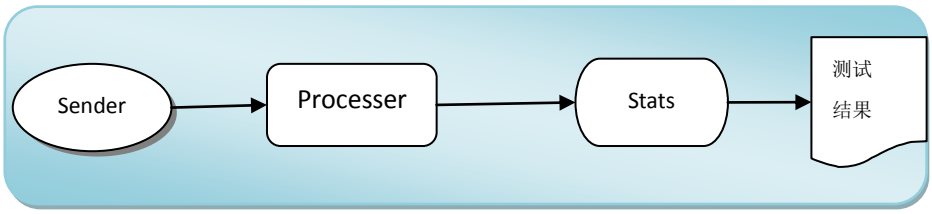
Storm 是一个流处理系统，它以 tuple 为基本单位，每个 tuple 可以包含多个字段（field）。我们给 tuple 定义两个字段：

- Data： 存放原始的数据，这里是 1000 字节的数据，此测试中我们仅仅是直接的转发数据，所以唯一的处理开销就是 1000 字节的内存拷贝
- tsInfo： 时间戳信息，每经过一个处理模块，我们就在此字段中追加上当时的时间戳，最后统计模块就可以根据这些时间信息计算出总延迟等。由于不同的机器时间戳并不同步，这给计算延迟带来了固有误差，解决的办法就是把数据发送模块和最后的统计模块放到一台物理机上。

关于在分布式集群上测试 storm 的一个说明：在 storm 上，我们很难给某个模块（component）指定其运行的物理机，storm 总是自动的把任务平均分配给集群中的各个机器，因此在测试中我们将使用 storm 的工作方式来扩展，而非设计非典型的情景（给某个 component 指定特定的机器来运行，从而打破这种平均分配原则）。

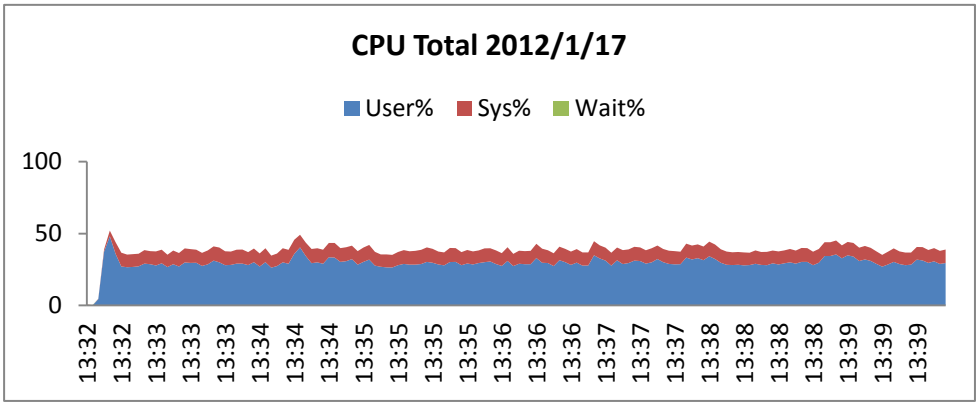
各测试以及结果

a) 在单个主机上，采用如下的方式构造拓扑

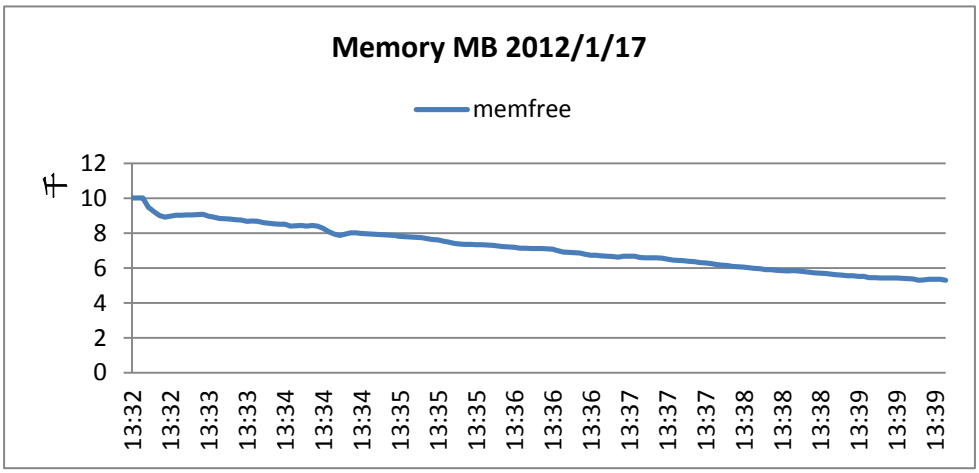


系统资源的利用率如下

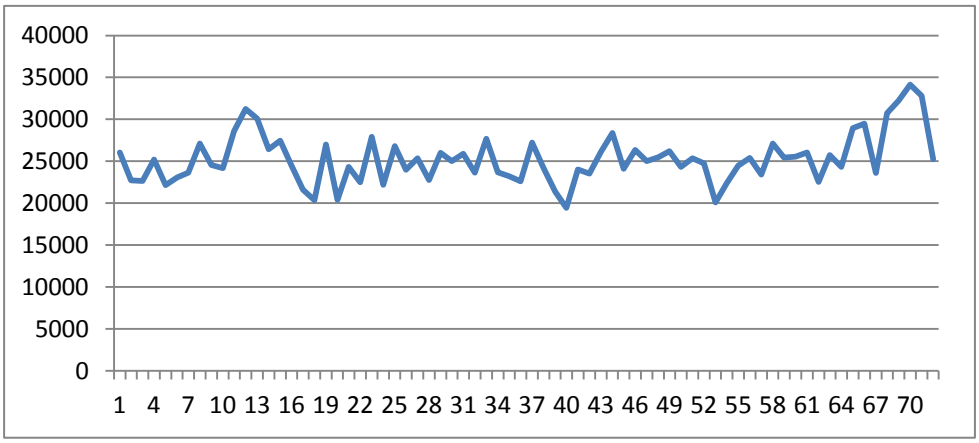
CPU 利用率



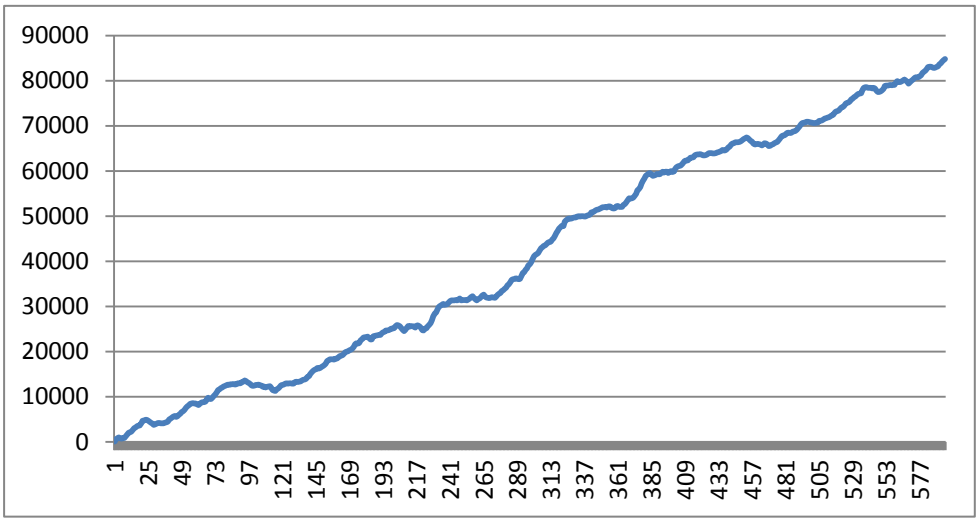
内存使用情况



吞吐量：平均值为 **25253** 条/秒



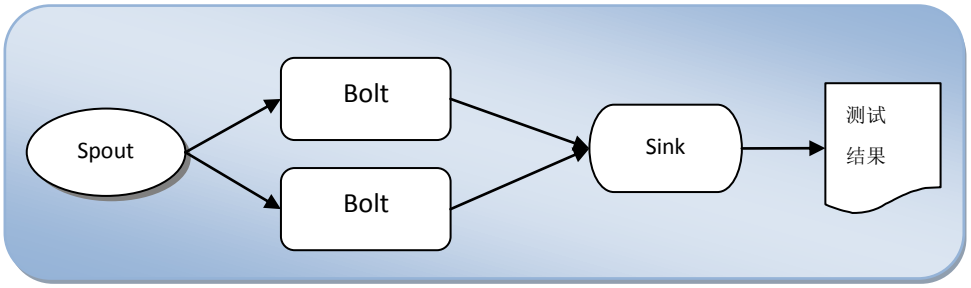
数据处理的延时如下图所示，（单位 毫秒）



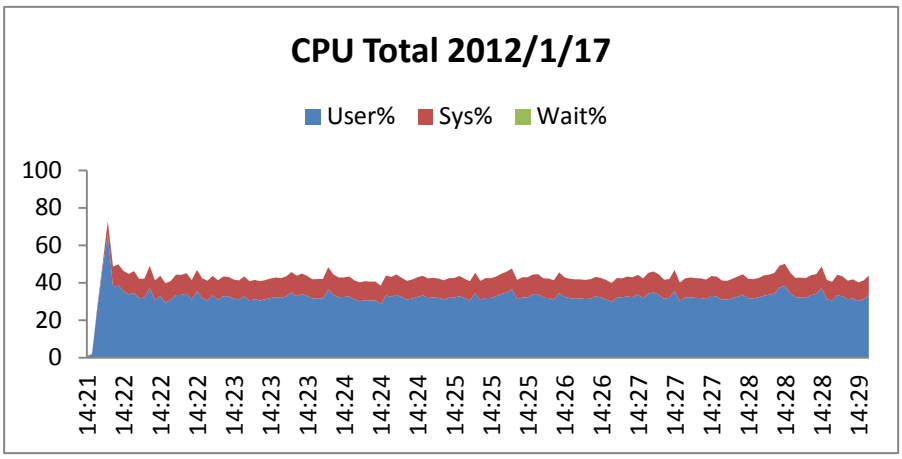
数据处理延迟越来越大的原因是，后端数据处理模块处理的速度较慢，所以数据发送端发送的数据产生了累积，并且累积量越来愈大，所以延迟的值就越来越大。较精确的测试处理延迟方案见（测试 e）

b) 由上面的测试可知，processer 模块的处理速度跟不上 sender 的发送速度，导致数据累积在发送端，本测试用例并行扩展 processer，查看并行扩展以后是否可以提高处理的速度从而使得 sender 模块没有数据累积。

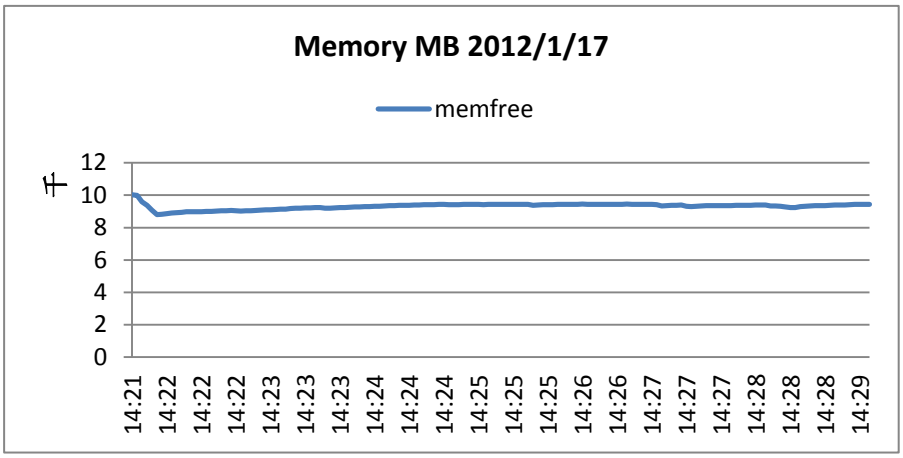
i. 拓扑图如下：



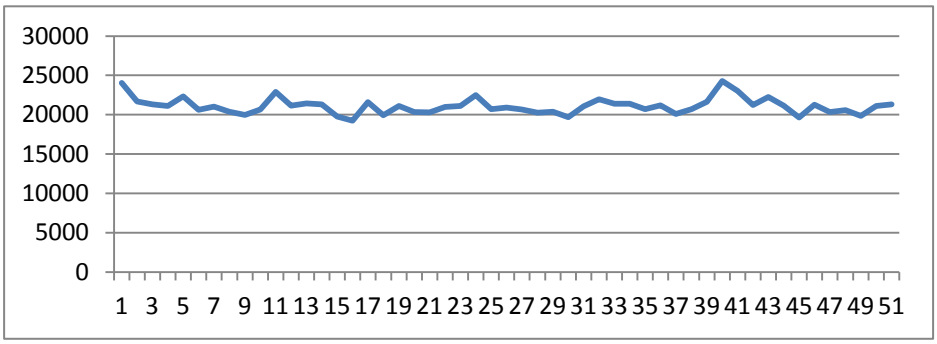
ii. 系统资源的使用情况如下：



内存利用率



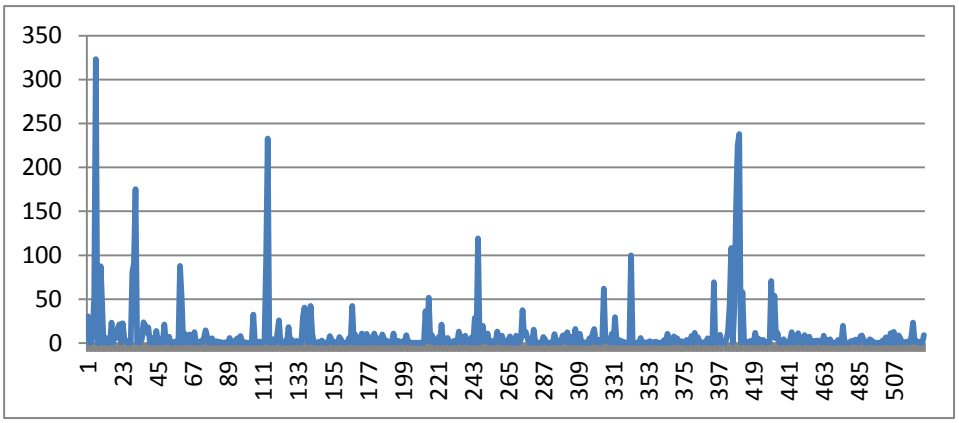
iii. 系统的吞吐量如下图所示，平均值为 **21809** 条/秒



在本测试用例中 sender 每秒钟发送大约 22000 个 tuple。

为什么并行以后系统的吞吐量还下降了呢？这是由于系统中增加了处理负载，sender 的每秒钟发送的数据减少了，由下图的“处理延迟”可见 sender 端基本上不累积数据了。

数据处理延迟（单位 毫秒）：

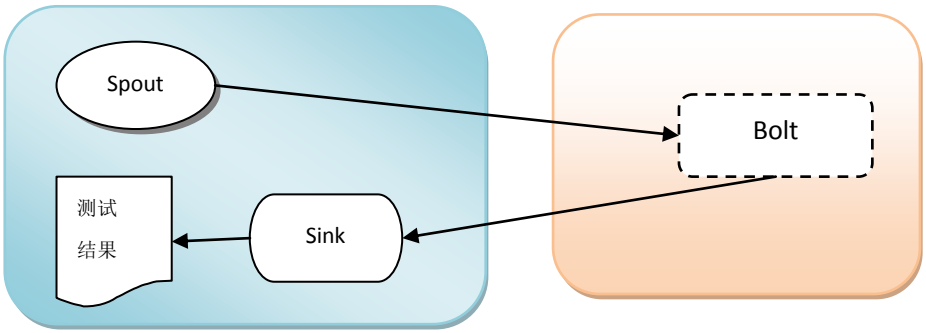


绝大部分的处理延迟就在毫秒级，偶尔有一些处理延迟较大，分析认为是由于调度器对多个任务的调度、以及 sender 线程偶尔获得较大的资源增大了发送能力导致的。数据处理延迟没有线性增长，说明 sender 发送的数据都可以被实时处理掉（或有较小延时）。

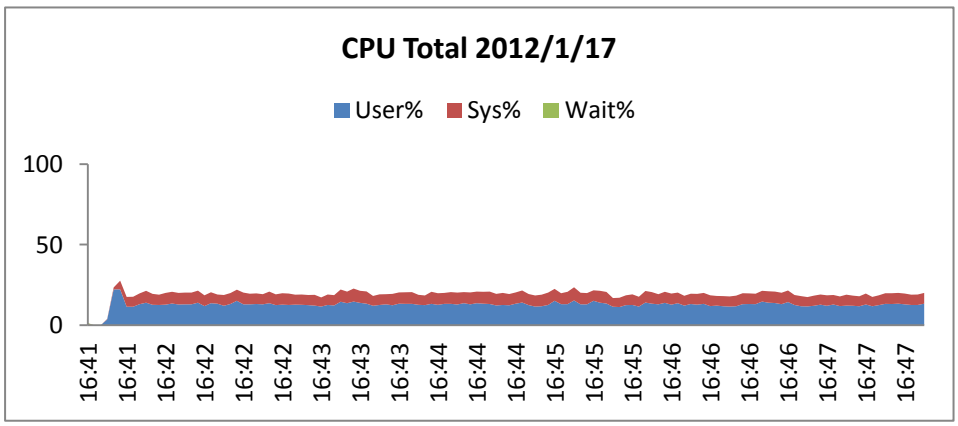
c) 在集群上的扩展测试，

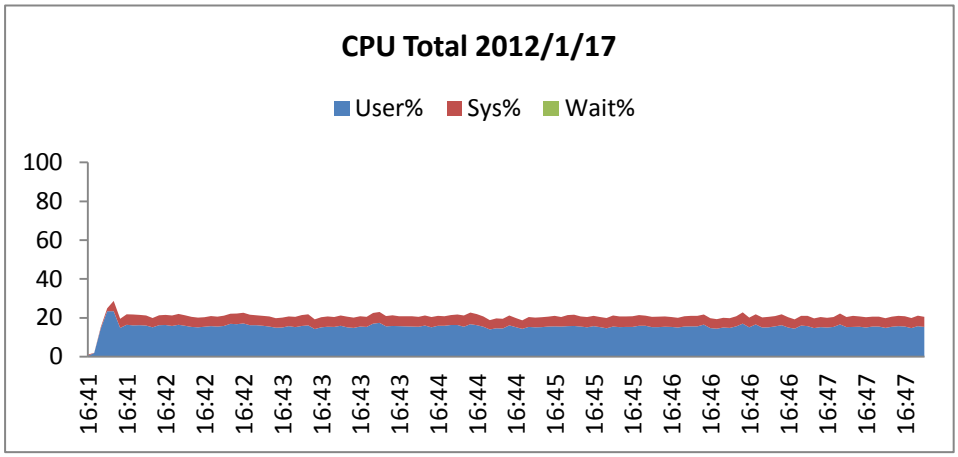
sender 与 processer 不在同一台机器，并与以上测试结果对比。

由于不同主机上时间戳不同步，为了消除由此带来的误差，我们必须将数据产生模块 sender 和最后的计算模块 stats 放到同一台计算机上，将数据处理模块放到另一台计算机上，如下图所示

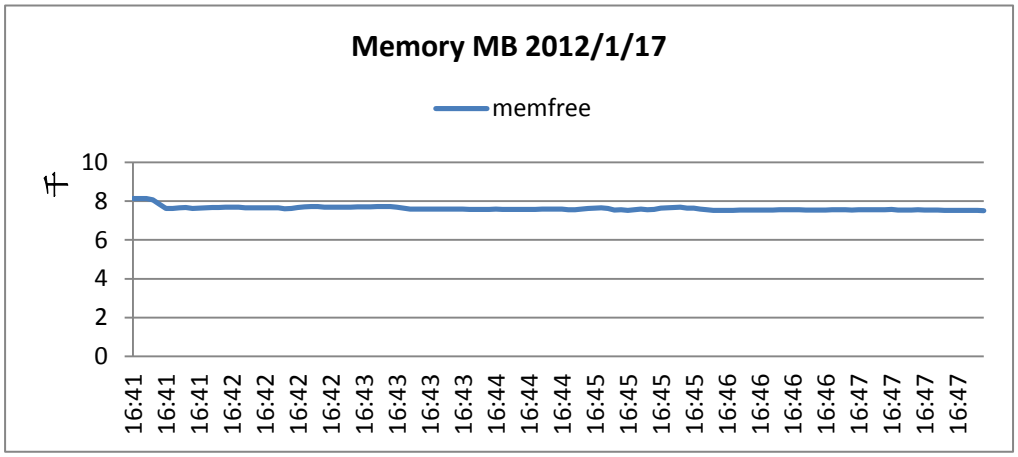


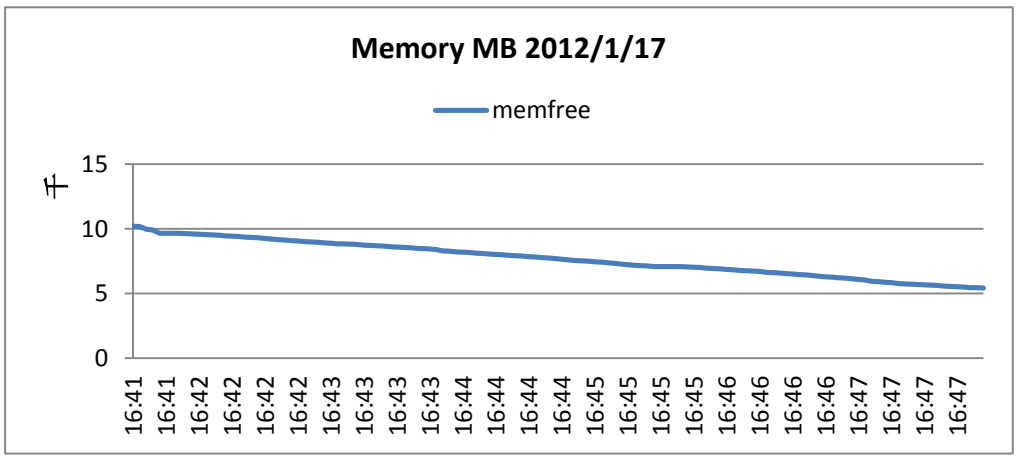
系统资源的利用率如下图所示：



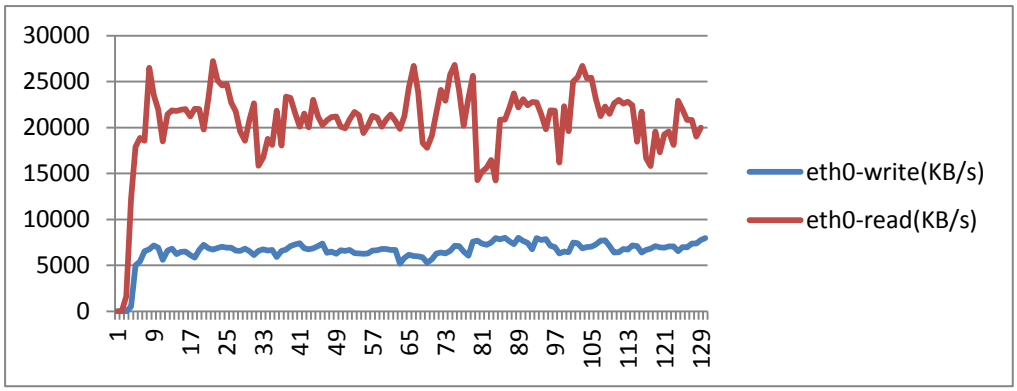


内存的使用情况

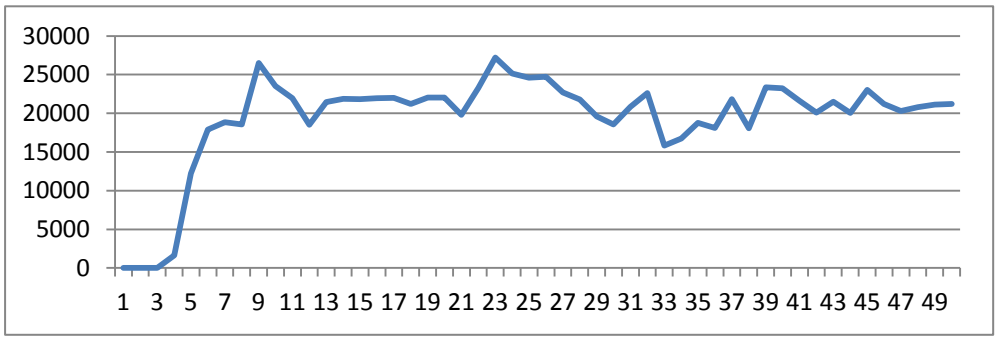




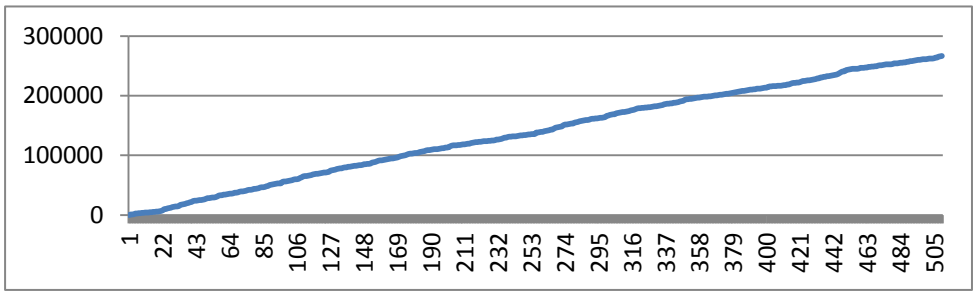
网络利用率



吞吐量如下图，平均值为 **20771** 条/秒

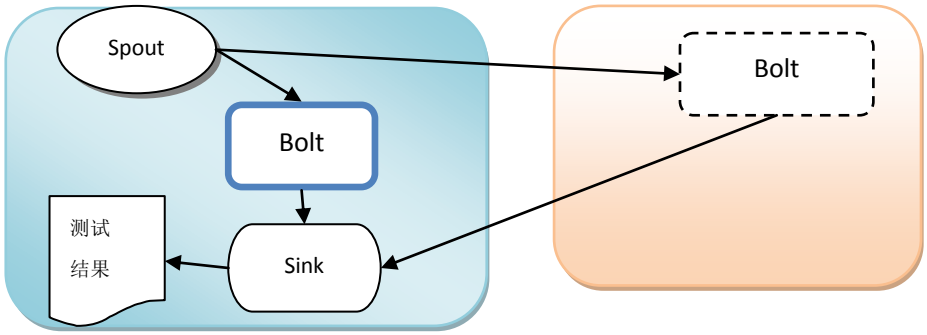


处理延迟如下图所示 （单位 毫秒）：

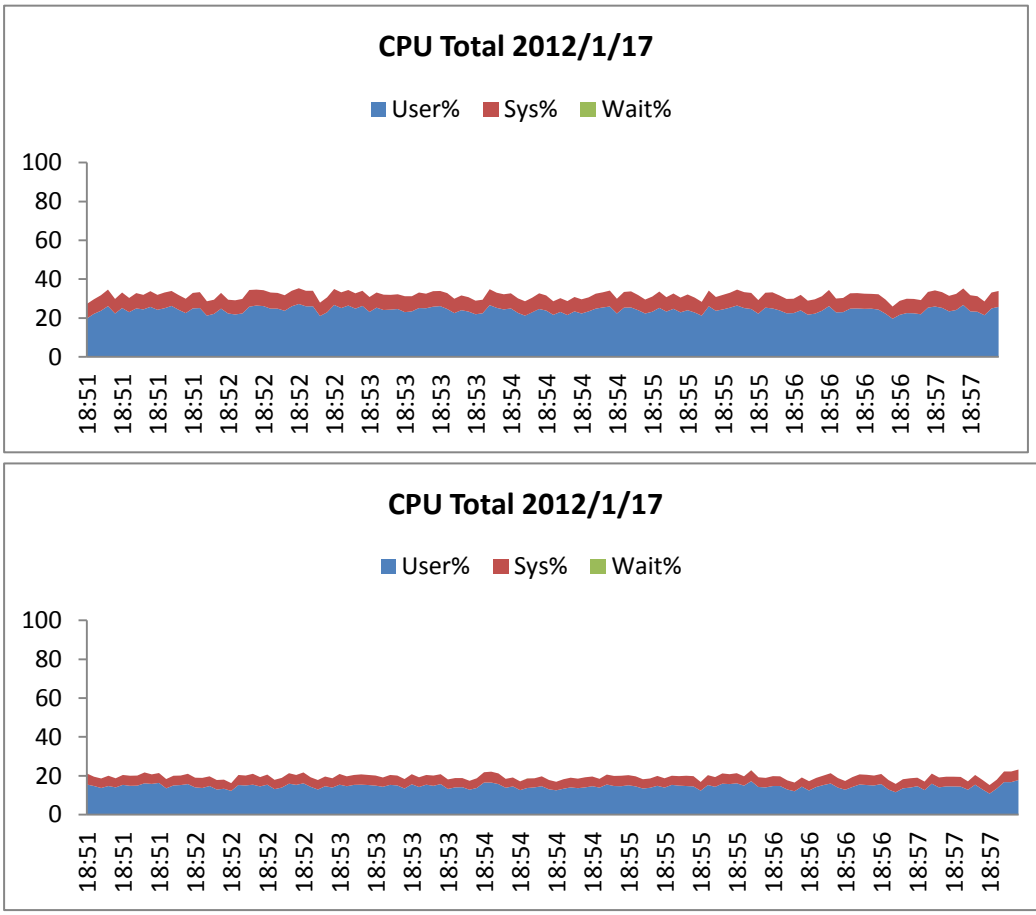


由于 sender 线程发送数据较快，数据会不断累积，所以数据延迟会不断增大，这说明后端的处理能力不足，即数据发送端产生数据过快（较精确的测试处理延时方案见测试（e））

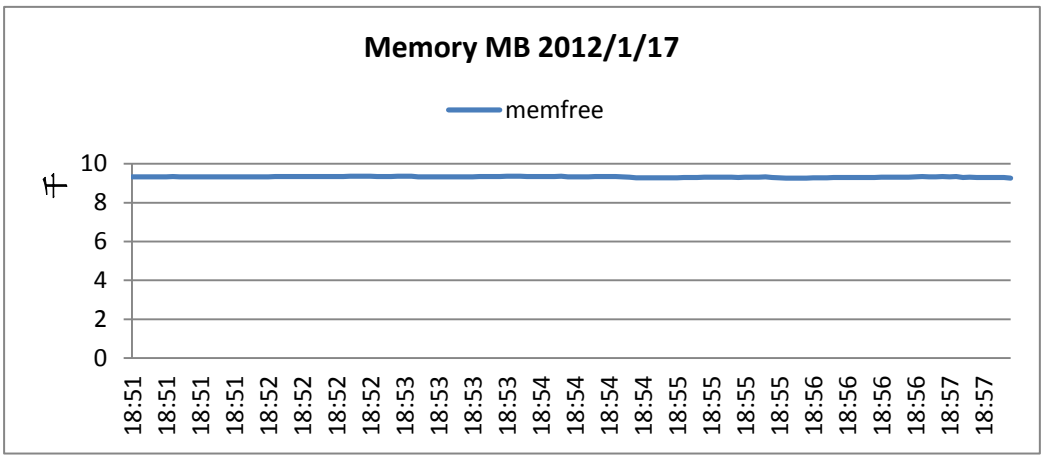
d) 横向扩展，增加处理模块，消除数据产生段的累积

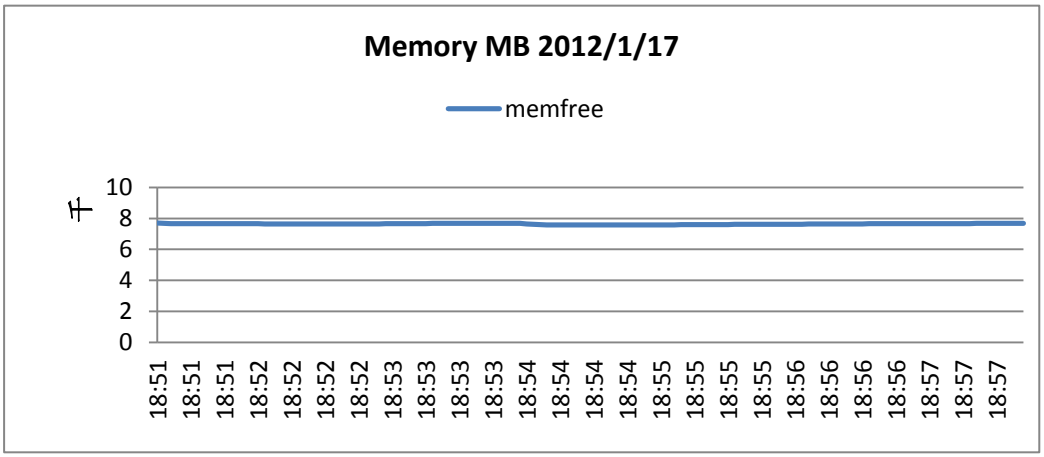


CPU 的利用率如图

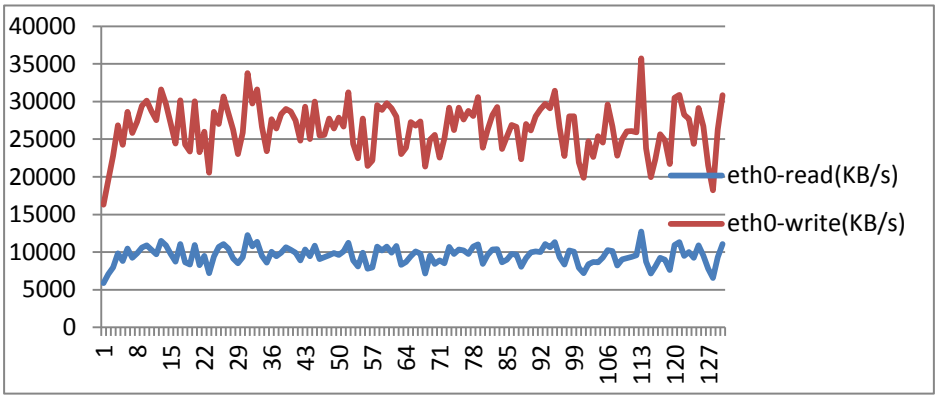


内存使用情况如图



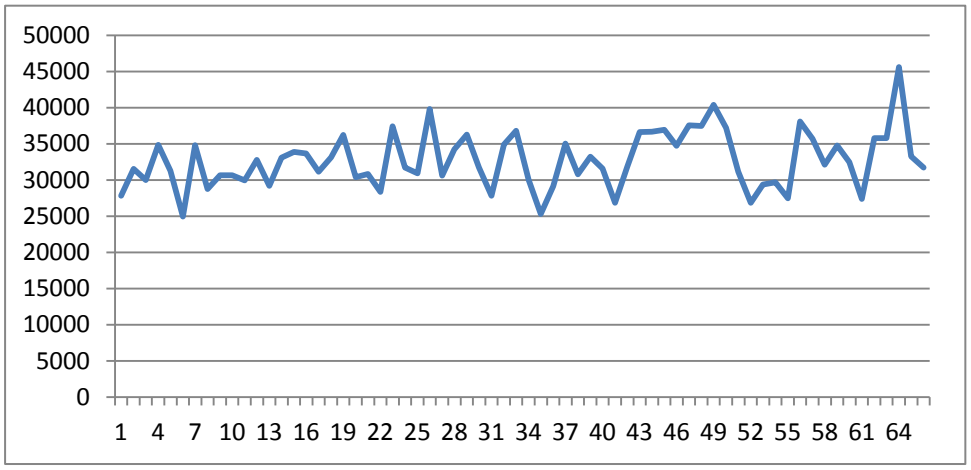


网络情况如图：

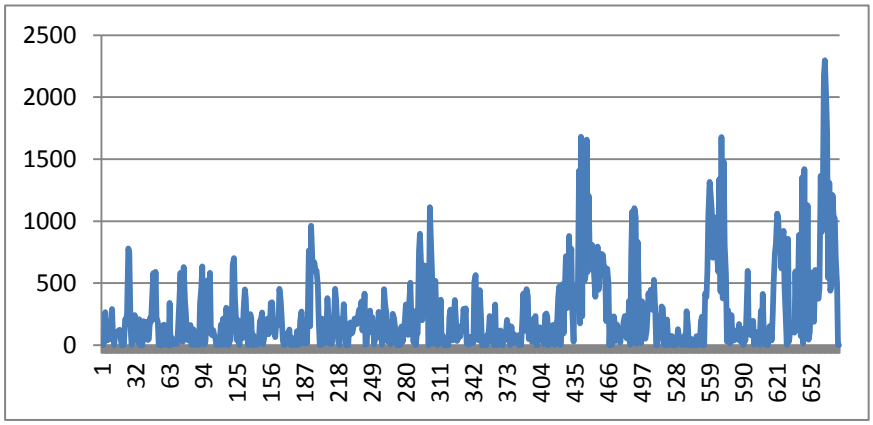


测试结果以及分析：

系统吞吐量：（均值为 32701 条/秒）



数据处理的延迟如下所示，（单位为毫秒）



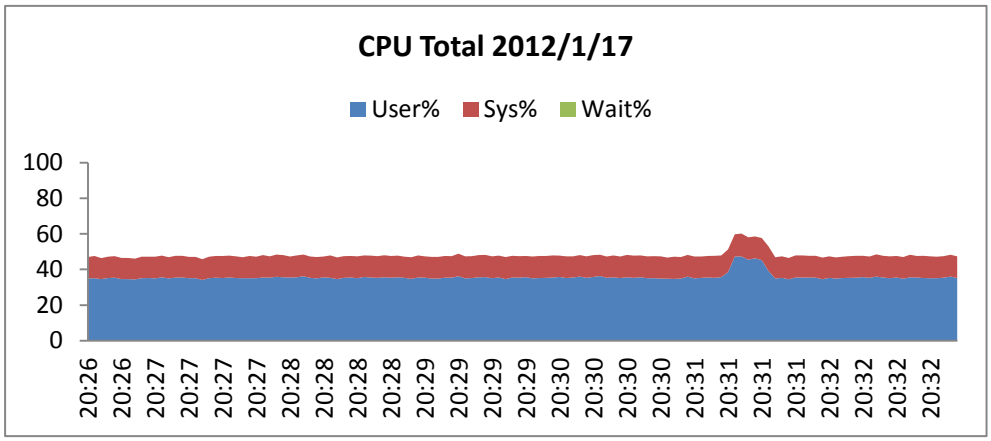
数据处理的延迟大部分在毫秒级，也有少数的延迟较大（几百毫秒），分析认为是由于任务调度等引起的 sender 线程在某一时刻拥有了较多的资源，从而发送了较多的数据，但是后端的处理能力较强可以随时将这些数据处理完毕，从而 sender 模块不会累积太多的数据，即 sender 发送的数据可以全部被处理。

e) 在数据高可靠情况下测试性能

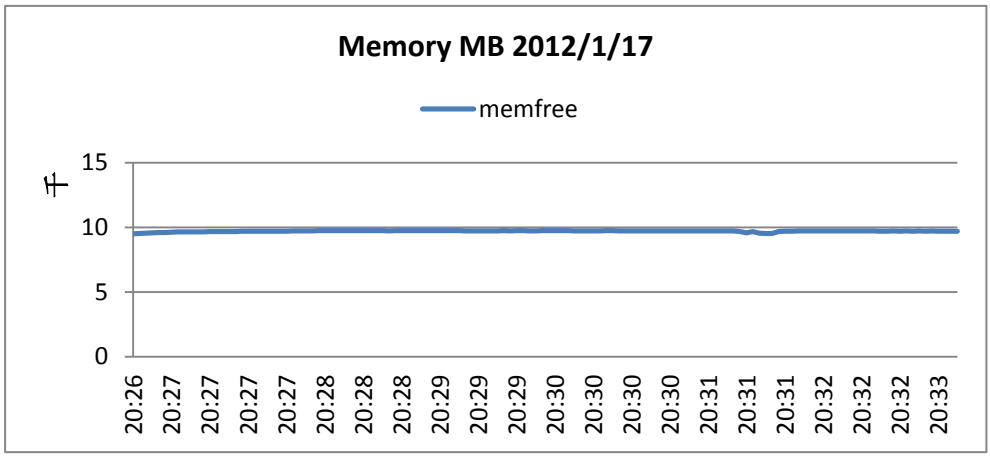
本测试在单节点上运行，拓扑原理与 a)相同，不过每个 tuple 都需要被 ack，本测试中还限制了 sender 中未被 ack 的数据不能多于 50 条。由于系统中增加了 ack tuple，数据量将至少增加一倍，所以有效数据处理性能将下降。

资源使用情况：

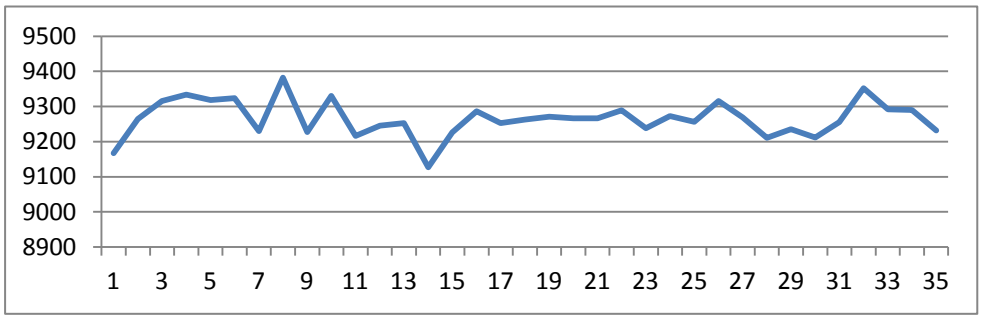
CPU 利用率



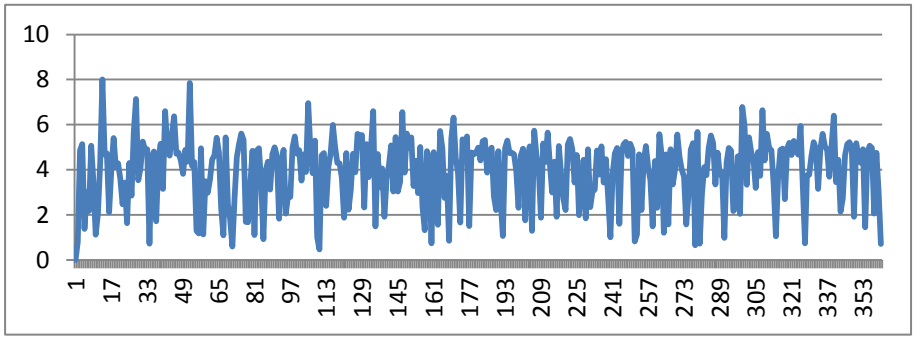
内存使用情况



吞吐量， 约为 9250 条/秒



数据处理的延迟见图 （单位 毫秒）

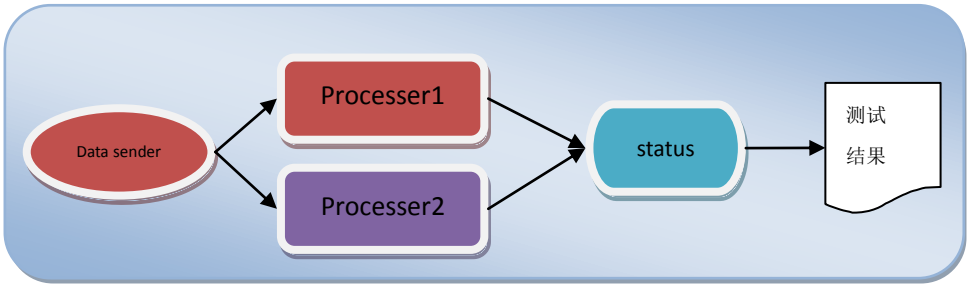


结果分析，由于我们限制 sender 发送的速度（未被 ack 的 tuple 不能超过 50 个），所以它发送出来的 tuple 都能被及时的处理，由上图可见，**所有 tuple 的处理延迟都在毫秒级，这个可以认为是 storm 系统本省的处理延迟**。

f) Java GC 对 tuple 处理延迟的影响

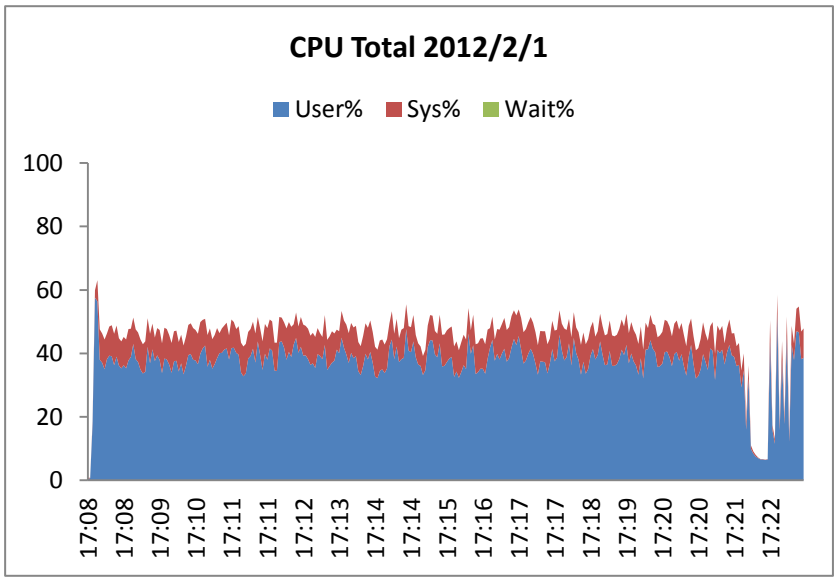
i. java 在回收垃圾内存的时候会停止当前正在运行的程序，本测试用例测试 java gc 对系统性能（尤其是处理延迟）的影响。为了突出 gc 对系统的影响，我们将两个处理单元（datasender 与 processer1）放到一个 worker（即 jvm）中，从而使得内存回收的问题更加突出和易于观测。
（**注： 在以上的其它测试中，每个处理单元各占一个 worker（jvm），所以未出现这里测试的结果**）

ii. 测试原理图同测试用例 b)

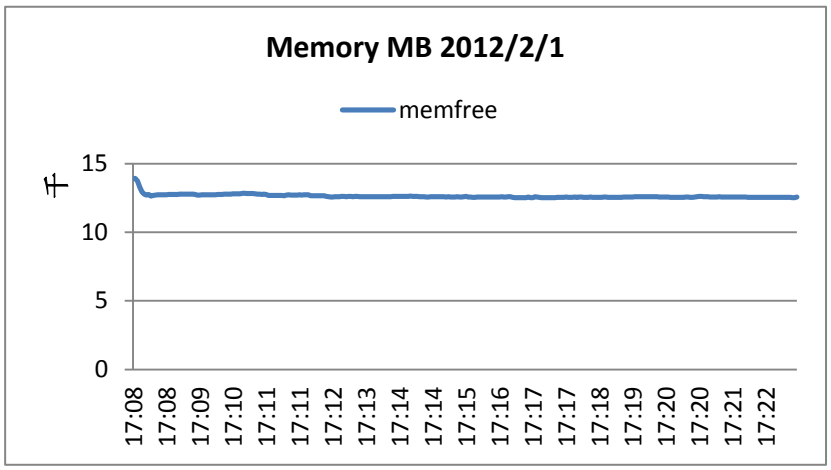


iii. 测试结果如下：

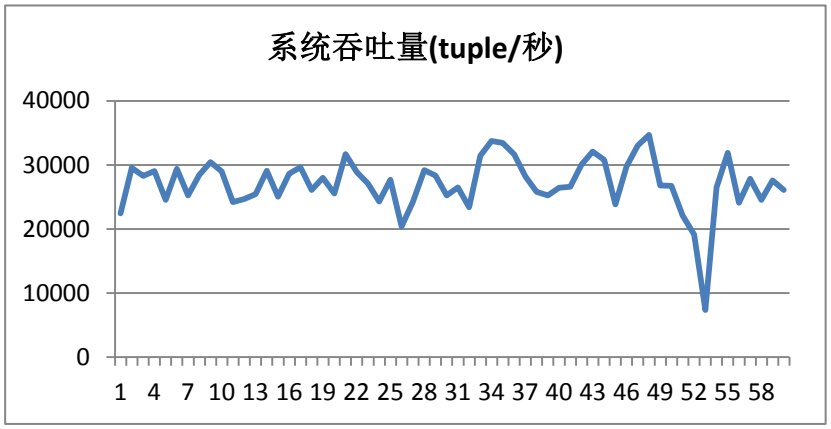
1. CPU 利用率



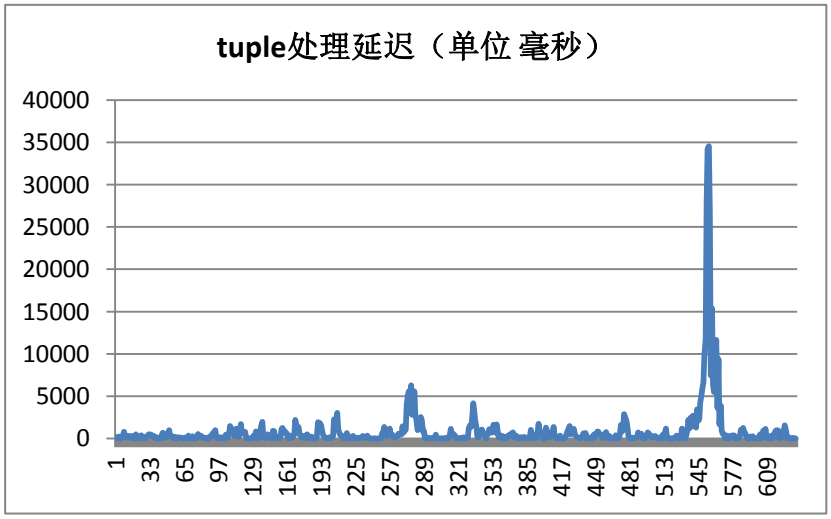
2. 内存的使用情况



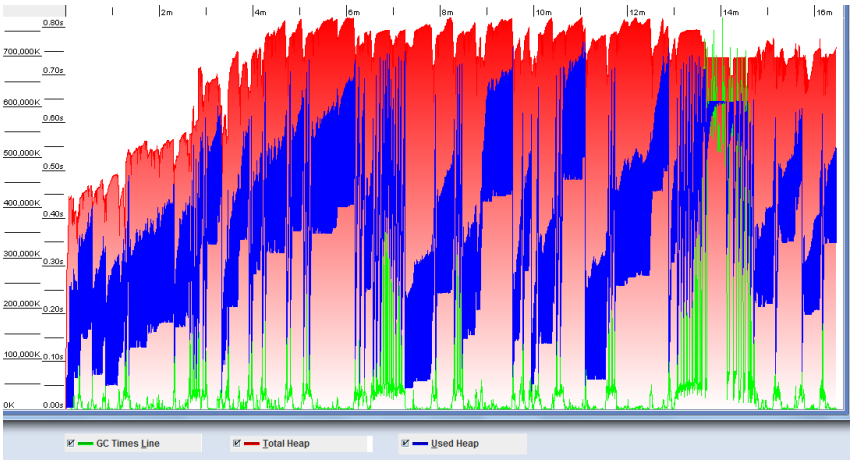
3. 系统吞吐量



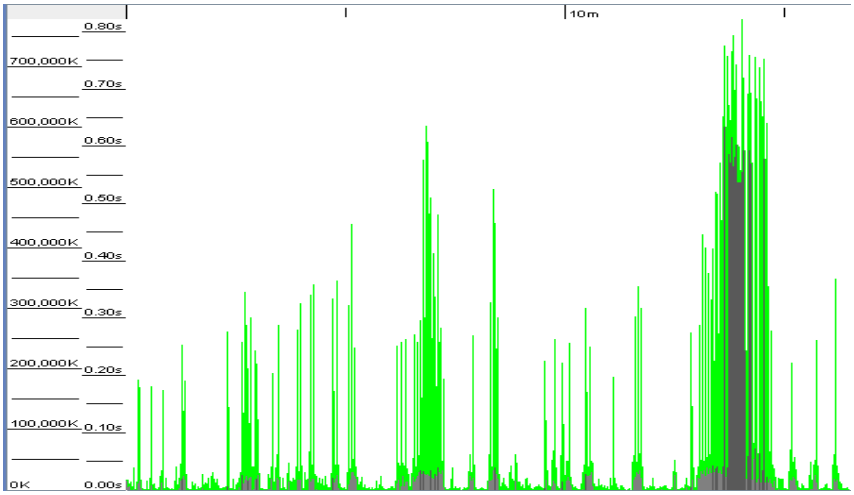
4. 各 Tuple 处理延迟



5. Java 虚拟机内存使用情况统计图



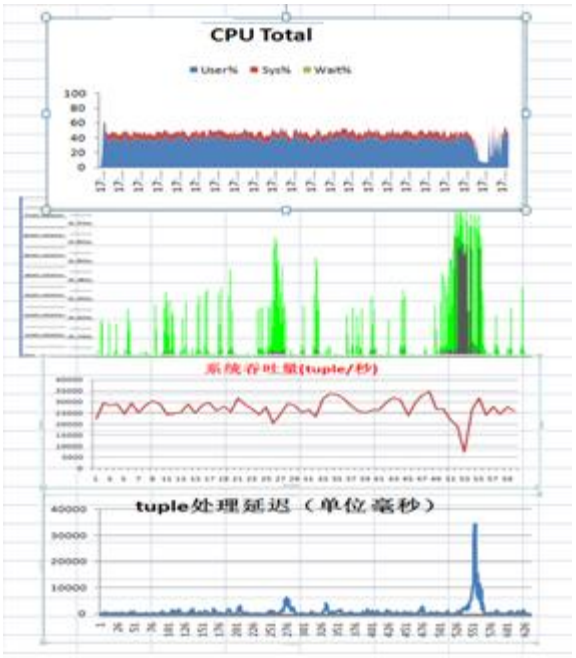
6. Java GC 垃圾回收模块运行时间统计图



Java GC 的数据指标如下：

系统运行时间	16m8s
full gc 时间	63.22s（49.2%）
增量 gc 时间	65.39s （50.8%）
最小暂停时间	0.00037s
最大暂停时间	0.82217s

7. 为了便于分析，我们将 cpu 利用率、tuple 处理延迟、系统吞吐量和 gc 合并到用一张图中



对上图前半段的分析：java gc 对 tupe 处理延迟的影响不是特别明显，一般在几百毫秒左右 （延迟图中较小的波动）。当每个 worker （jvm）中只包含一个处理单元时（内存使用较小且可被及时回收），测试结果基本与此相同（小波动，没有 gc 密集回收垃圾的过程）。

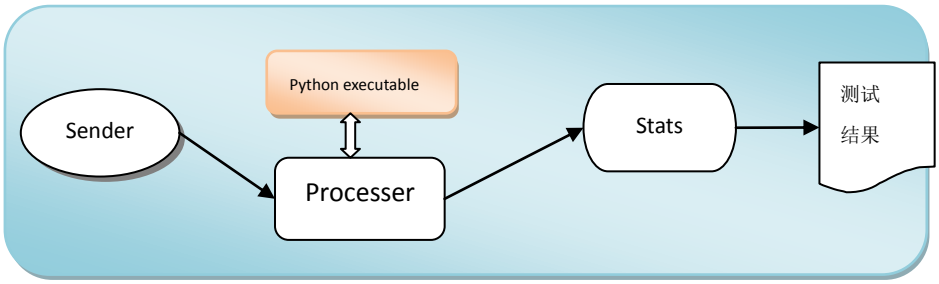
对上图“tuple 处理延迟中”较大脉冲的分析： 由于两个耗费内存的处理模块放到了同一个 jvm 中，所以此进程的内存使用量很大，当 GC 不能回收足够的内存时(发送端有累积的 tuple 不能被及时处理,它们占用内存)，大部分的时间都会耗费在稍后的内存回收，由上图可见，CPU 的利用率骤降，原因是 gc 运行时间增加，jvm 中被停止的进程长时间不能得以运行，进而导致 tuple 的处理延迟增加。这个结果不是每次测试都能出现，只有当某些原因导致发送端出现 tuple 累积时才会出现。即使未出现此现象也说明系统有潜在的较大延迟风险,除非设计中已经保证数据处理单元有足够的能力来处理发送端发送来的数据。

综上所述，java gc 对系统的影响可以分两种情况阐述： 1）内存足够（内存可被及时回收）的情况下，gc 对象影响不大 2）jvm 中内存使用接近上限且暂时不可回收时，gc 对系统影响极大。所以，storm 在设计、开发时需要仔细考量各处理单元内存的使用以及系统中 worker 的数量。

g) Storm 使用外部处理程序时的性能

本测试用例主要测试使用外部处理程序的情况下，系统的整体性能。使用外部处理程序的时候，storm 将外部处理程序作为子程序来运行，并使用 Json 格式来交换数据。本测试中我们使用 python 脚本作为外部处理程序。

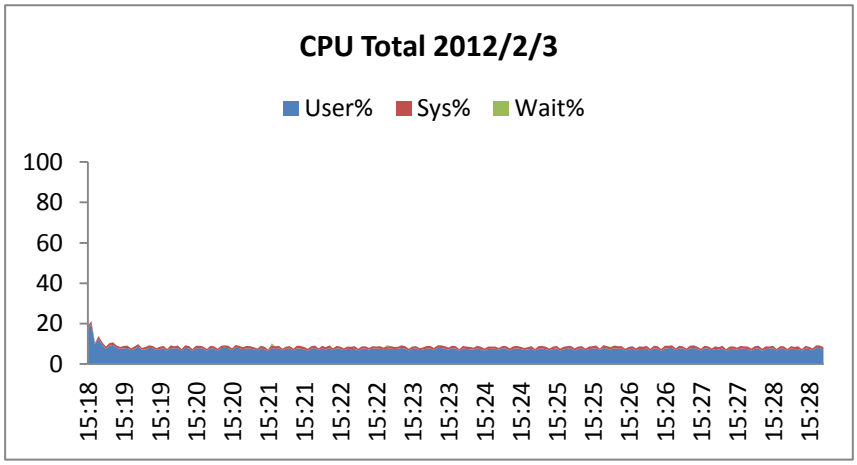
i. 测试原理图如下：



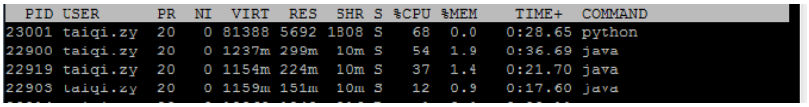
测试中，sender，processer 等都是单节点的，所以本测试结果为单条处理线的处理能力。

ii) 测试结果：

1) CPU 利用率

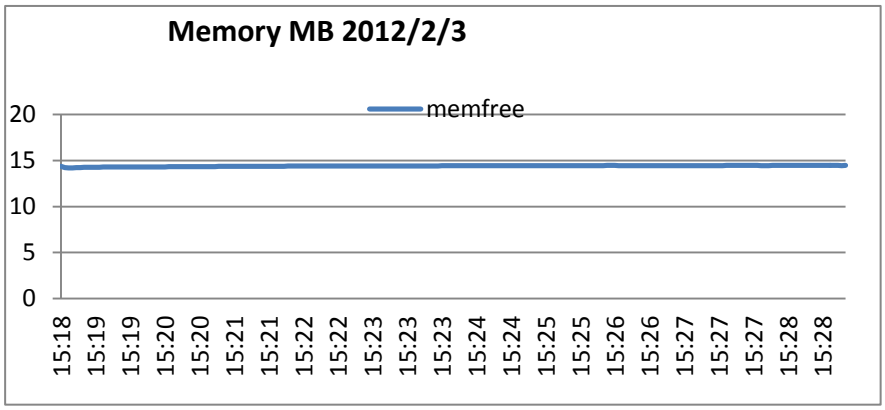


2) 各进程 CPU 利用情况

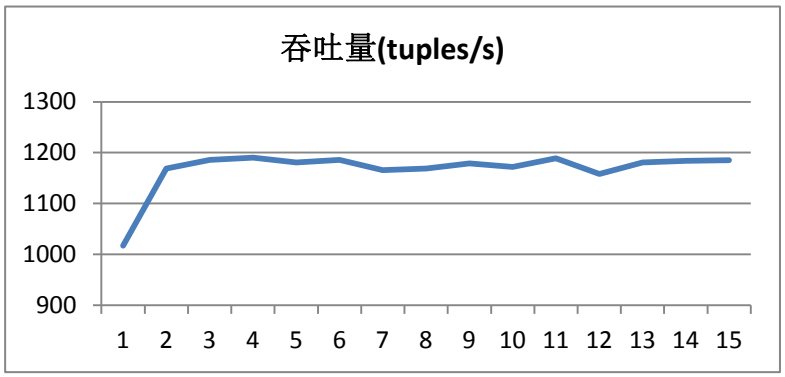


PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
23001	taiqi.zy	20	0	81388	5692	1808	S	68	0.0	0:28.65	python
22900	taiqi.zy	20	0	1237m	299m	10m	S	54	1.9	0:36.69	java
22919	taiqi.zy	20	0	1154m	224m	10m	S	37	1.4	0:21.70	java
22903	taiqi.zy	20	0	1159m	151m	10m	S	12	0.9	0:17.60	java

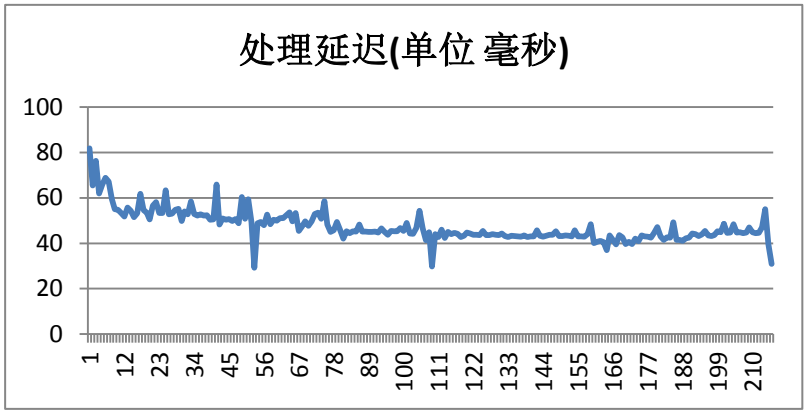
3) 内存使用情况



4) 吞吐量（tuples/s）



5) Tuple 处理延迟



测试结果分析：由上面的测试可见，使用外部处理程序时，系统的处理能力只有 1000 tuple/s，性能下降明显。 分析认为性能陡降的原因有二： 1） 所有的 tuple 都经过 Json 格式与外部程序交互，格式转换的过程耗费了 CPU circle；2） storm 把外部处理程序当做子进程，使用 linux 管道来通信，由于 linux 管道(pipe)使用的是 4K 大小的页面做中转，所以在数据量较大的时候会有性能的损耗，测试中每个消息至少为 1K bytes，所以很快就会将 pipe 使用的内存用完儿产生等待。增加外部程序个数（即 processer 处理单元的并行度，不超过系统中 CPU 的个数），性能基本上有线性的提升。

由处理延迟的测试结果可见，使用外部处理程序的时候，tuple 处理延迟比使用 storm 内建的处理机制要大十倍左右。

测试结论

经过上面的测试我们可以得出以下的结论：

- storm 单条流水线的处理能力大约为 20000 tupe/s, (每个 tuple 大小为 1000 字节)
- storm 系统本省的处理延迟为毫秒级
- 在集群中横向扩展可以增加系统的处理能力，实测结果为 1.6 倍
- Storm 中大量的使用了线程，即使单条处理流水线的系统，也有十几个线程在同时运行，所以几乎所有的 16 个 CPU 都在运行状态，load average 约为 3.5
- Jvm GC 一般情况下对系统性能影响有限，但是内存紧张时，GC 会成为系统性能的瓶颈

- 使用外部处理程序性能下降明显，所以在高性能要求下，尽量使用 storm 内建的处理模式