注意，Sink 处理器与 Sink 运行器不同。Sink 运行器实际上是运行 Sink 的，而 Sink 处理器决定了哪个 Sink 应该从自己的 Channel 中拉取事件。当 Sink 运行器要求 Sink 组告知其中一个用于从 Channel 中拉取事件的 Sink，且将事件写到下一阶段（或存储系统）时，Sink 处理器就是实际用来选择完成这个过程的 Sink 组件。Flume 自带了两类 Sink 处理器：*load-balancing sink* 处理器和 *failover sink* 处理器。

159 对于指定的 Sink 组的一部分，Sink 处理器使用 `processor.type` 后缀进行配置。配置可以使用 `processor.` 前缀传递给 Sink 处理器。这里是如何配置的一个例子。

```
agent.sinks = s1 s2
agent.sinkgroups = sg1
agent.sinkgroups.sg1.sinks = s1 s2
agent.sinkgroups.sg1.processor.type = load_balance
agent.sinkgroups.sg1.processor.backoff = false
```

## Load-Balancing Sink 处理器

假设你有一个拓扑结构，在推送数据到 HDFS 之前，其中第一层从成千上万的应用程序服务器接收数据，第二层通过 Avro RPC 从第一层接收数据。为简单起见，我们假设第一层有 100 个 Agent，第二个层有 4 个 Agent。在尽可能简单的拓扑中，第一层每个 Agent 将有四个 Avro Sink 用来推送数据到第二层的每个 Agent。该工作正常运行，直到其中第二层的一个 Agent 失败。此时，配置发送数据的 Sink 将不会发送任何数据，直到第二层失败的 Agent 重新上线。

除了该 Sink 耗尽了 Agent 上的几个线程这样一个事实（一个用于 Sink Runner，另一个用于使用 Netty 发送数据的线程池）浪费了 CPU 周期，直到第二层 Agent 启动并运行，通过创建删除事件的事务且回滚，该 Sink 也会给 Channel 造成额外的压力。对于 File Channel，尽管事务没有被提交，许多将要写入文件的读取操作（即使事务没有提交，这些读取操作也要写入文件），也造成了 I/O 成本和磁盘空间成本。如图 6-2 所示。