(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 103678085 A
(43)申请公布日 2014.03.26

(21)申请号 201310688931.0

(22)申请日 2013.12.16

(71)申请人 上海证券交易所
地址 200120 上海市浦东新区浦东南路 528 号

(72)发明人 金鑫 武剑锋 王泊 刘凯 陈雷 王程程 惠敏顺 黄寅飞 白硕

(74)专利代理机构 上海三方专利事务所 31127
代理人 吴干权 李美立

(51)Int.Cl.
G06F 11/30(2006.01)

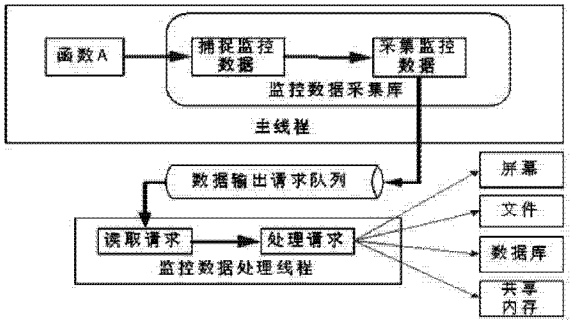
权利要求书2页 说明书6页 附图2页

(54)发明名称

一种流量动态可控的系统监控数据采集方法

(57)摘要

本发明涉及数据采集和流量控制领域，具体是一种流量动态可控的系统监控数据采集方法，所述的流量动态可控的监控数据采集方法主要分为三个部分，分别是监控数据异步采集处理，流量控制开启检测和流控数据合并机制；所述的监控数据异步采集处理方法如下：通过调用系统线程接口，监控模块在初始化后启动一个单独的监控数据处理线程负责系统数据的多目标处理输出；动态流控的一个总周期可以划分为两个阶段：流控检测阶段 T_d 和流控执行阶段 T_e。本发明同现有技术相比，其优点在于：通过流速权控制算法，保证关键实时系统的平稳运行，能够有效地满足不同监控目标的监控数据采集需求，并通过特征数据的合并方法，保证了落地数据的质量。



CN 103678085 A

1. 一种流量动态可控的系统监控数据采集方法，其特征在于所述的流量动态可控的监控数据采集方法主要分为三个部分，分别是监控数据异步采集处理，流量控制开启检测和流控数据合并机制；

所述的监控数据异步采集处理方法如下：通过调用系统线程接口，监控模块在初始化后启动一个单独的监控数据处理线程负责系统数据的多目标处理输出；单独的监控数据处理线程不停轮询数据输出请求队列，一旦发现有新的请求时，就从队头弹出数据记录并执行真正的输出请求，一直到将队列中的记录处理完毕；

所述的流量控制开启检测方法如下：监控数据采集库的采集接口在进行数据入队操作时，会实时检查系统监控事件的发生频率，并动态调整采集数据的流量，从而避免大量数据操作对系统整体性能的冲击，流控算法首先在流控检测周期内计数到达队列的数据请求数目，一旦该请求数达到或超过请求数阈值，就立即开启流量控制机制，并一直持续到流控执行周期的时间段结束。因此，动态流控的一个总周期可以划分为两个阶段：流控检测阶段 T_d 和流控执行阶段 T_e；

所述的流控数据合并机制方法如下：在流控执行周期，监控数据采集库的采集接口不再将新到的数据加入数据输出请求队列，而是通过初始化时的哈希表进行相同特征数据的合并，从而降低实际数据流量的目标，数据的合并机制是：对于未出现过的特征数据类型，在哈希表里面增加一条；否则，只增加相同记录类型的计数。

2. 如权利要求 1 所述的一种流量动态可控的系统监控数据采集方法，其特征在于所述的单独的监控数据处理线程的数据输出工作为屏幕打印或进行磁盘文件写入或通过网络传输数据至远程的数据库服务器。

3. 如权利要求 1 所述的一种流量动态可控的系统监控数据采集方法，其特征在于所述的流控检测周期设为 0，表示始终不执行流控；如果数据输出请求数阈值设为 0，表示始终执行流控机制。

4. 如权利要求 1 所述的一种流量动态可控的系统监控数据采集方法，其特征在于所述的监控数据异步采集处理方法需在采集模块初始化时选定异步数据输出请求队列长度，队列长度过小会导致大量监控数据无法入队而被丢弃，所述的异步数据输出请求队列长度选定方法如下：

监控事件数据的到达和监控事件数据的落地之差即为该时间段的队列长度的变化量 Δ，在上一个交易日 D_{T-1}，令 e_i 表示第 i 秒内发生的事件数，对每条采集的数据定义了 m 个输出目标，每个输出目标可以通过性能测试数据的平均值获得其单条数据处理时间的期望值 $\bar{\mu}$，那么单位时间内 m 个目标的总服务速度(可以处理的数据)为 $v_m = 1 \Big/ \sum_{i=1}^{m} \bar{\mu}_i$，综上，第 i 秒对队列的改变量为：

$$\Delta = e_i - v_m \quad (1)$$

设第 i-1 秒后的队列长度为 L_{i-1}，那么第 i 秒后的队列长度即为：

$$L_i = \max(L_{i-1} + \Delta, 0) \quad (2)$$

注意因为队列长度不可以为负数，所以在服务速度大于数据到达速度的场景下，队列最小为 0；

交易时段总长为 t，那么在 D_{T-1} 的整个交易时段内队列长度出现过的最大值便可以作为未来队列长度的基准值 L_q：

$$L_q = \left\lceil \alpha\left(\max_{i\in[1,t]}(L_i) \right) \right\rceil + c, (\alpha \geq 1, c > 0) \qquad (3)$$

其中 α 为调和系数，即对历史数据中最坏情况下的队列长度再进行一定系数的放大，以保证未来使用该基准值后，监控数据被丢弃的概率很小，c 为某个正整数常量，用于保证队列基准值的最小值，其选取值可以是任意一秒的非零事件数 e_i（$e_i > 0$）。

5. 如权利要求 1 所述的一种流量动态可控的系统监控数据采集方法，其特征在于所述的流量控制具体方法如下：

将一个流控周期 T_w 划分为流控检测阶段 T_d 和流控执行阶段 T_e，定义 $\lambda = m/T_d$，其中，λ 为数据到达速度的上限阈值，m 为数据输出请求数阈值，数据到达速度超过该阈值时，认为磁盘读写会对系统性能产生不良影响，对于相同的 λ 值，m 设置的越低，T_d 就越低，即在一个周期内会越早进入流控执行阶段，系统总体性能的损耗会越少；另一方面，越早进入流控执行阶段，就有越多的监控信息被压缩合并，对输出结果的质量产生削弱。

一种流量动态可控的系统监控数据采集方法

[技术领域]

[0001] 本发明涉及数据采集和流量控制领域，具体是一种流量动态可控的系统监控数据采集方法。

[背景技术]

[0002] 系统状态数据采集模块可以实时记录程序的运行情况、错误的发生现场等数据，并输出采集的数据到多个目标如屏幕，文件和数据库，从而帮助开发人员和运维人员了解系统状态、审计操作流程、排查错误原因、存档系统数据等。因此，在关键实时系统的开发和运行中，特别是对于具有高可用性目标的交易系统来说，数据采集模块的良好设计与实现不仅有利于代码重用，而且对于系统的稳定高效运行具有重要的意义。

[0003] 当今主流的监控数据采集框架或多或少都借鉴了 Log4j 库的设计理念。Log4j 实现了一个基于 Java 的数据记录工具框架，在 Log4j 中定义了 3 个重要概念，分别是 Logger、Appender 和 Layout ：Logger 用于定义不同的数据采集对象 ；Appender 描述输出流，如屏幕、文件、GUI 组件、Socket 服务器、NT 事件记录器、syslog 进程服务等，通过为 Logger 指定一个 Appender，可以区分不同对象的输出目标 ；Layout 用于指定监控信息的输出格式，通过为 Appender 来指定一个 Layout，可以决定监控信息以何种格式输出，比如是否有带有时间戳，是否包含文件位置信息等，通过这三个组件协同的工作，Log4j 为监控数据采集行为控制提供了极大的灵活性，此外，基于预定义的等级和过滤器，可以自由地选择需要记录的数据。

[0004] Log4j 虽然功能强大，但是该库只是面向 Java 语言的，因此基于 Log4j 机制的其他语言替代库相继应运而生，Log4c 便是模仿 Log4j 原理的 C 语言实现。类似地，它通过定义 Category，Appender 和 Layout 三个属性唯一确定一个数据采集对象，Log4c 不支持异步数据采集。

[0005] Log4g 同样是 C 语言的一个实现，它基于 glib 的 GObject 开发，支持异步数据输出，此外 Log4g 还支持面向 CouchDB 的数据记录，便于日后利用基于 NoSQL 环境的大数据分析。Log4cplus 和 Log4cxx 是 C++ 的 Log4j 实现，它们具有线程安全、灵活输出、以及多粒度控制的特点，但 Log4cplus 的移植性不高，而 Log4cxx 虽然通过使用 APR 增强了移植性，但是 APR 本身增加了系统的依赖性。此外，Zlog 作为一款比较新的数据采集库，通过定义不同的 fsync 阈值来实现记录监控数据落地请求和磁盘 IO 工作的异步处理。

[0006] 可以看到，现有的各种数据采集模块虽然能够实现监控信息的输出，但是在设计上都比较简单，缺乏针对系统状态变化时的动态流量调整，因此在应用系统的实时运行中，如果突然出现大量错误信息需要写文件等情况，会在一段时间内集中产生主机磁盘冲击，此时文件 IO 读写便会成为系统瓶颈，显著降低系统的整体性能。此外，部分数据采集模块（如 log4c）为了实现数据的及时输出，采用同步落地机制，那么在进行数据落地时会阻塞主逻辑的运行，应用进程需要等待 IO 输出完成才能继续后续代码的执行，于是在监控事件大量突发时，可能会因为磁盘 IO 吊住主程序的逻辑，从而对生产环境中实时程序的请求响

应产生灾难性的影响。

[0007] 综上，虽然目前有很多监控数据采集框架可供选择，但是关键实时系统的高稳定性和低延迟是重要的运行指标，如果由于监控逻辑的负载影响实时业务程序的运行和业务处理的及时响应，那么在设计上是有显著缺陷的。系统状态数据的采集可以帮助管理人员快速了解系统的运行情况、审计操作流程和排查错误原因，但是对于某些关键实时系统，如证券交易系统的稳定性和高可用性是衡量其设计运行水平的首要指标，那么在监控的事件（如系统错误等）频繁发生的情况下，大规模的数据采集和落地反而会对系统的整体性能和稳定性产生显著影响，严重时甚至会阻塞主应用程序的逻辑。

[发明内容]

[0008] 本发明的目的就是为了解决交易实时系统的系统状态数据的采集过程中，大规模的数据采集和落地会对系统的整体性能和稳定性产生显著影响的技术问题，实现在大规模数据采集和落地时既保证实时系统的平稳运行，又可以有效地满足特征数据的采集和分发工作的一种流量动态可控的系统监控数据采集方法。

[0009] 为了实现上述目的，发明一种流量动态可控的系统监控数据采集方法，所述的流量动态可控的监控数据采集方法主要分为三个部分，分别是监控数据异步采集处理，流量控制开启检测和流控数据合并机制；

[0010] 所述的监控数据异步采集处理方法如下：通过调用系统线程接口，监控模块在初始化后启动一个单独的监控数据处理线程负责系统数据的多目标处理输出；单独的监控数据处理线程不停轮询数据输出请求队列，一旦发现有新的请求时，就从队头弹出数据记录并执行真正的输出请求，一直到将队列中的记录处理完毕；

[0011] 所述的流量控制开启检测方法如下：监控数据采集库的采集接口在进行数据入队操作时，会实时检查系统监控事件的发生频率，并动态调整采集数据的流量，从而避免大量数据操作对系统整体性能的冲击，流控算法首先在流控检测周期内计数到达队列的数据请求数目，一旦该请求数达到或超过请求数阈值，就立即开启流量控制机制，并一直持续到流控执行周期的时间段结束。因此，动态流控的一个总周期可以划分为两个阶段：流控检测阶段 T_d 和流控执行阶段 T_e；

[0012] 所述的流控数据合并机制方法如下：在流控执行周期，监控数据采集库的采集接口不再将新到的数据加入数据输出请求队列，而是通过初始化时的哈希表进行相同特征数据的合并，从而降低实际数据流量的目标，数据的合并机制是：对于未出现过的特征数据类型，在哈希表里面增加一条；否则，只增加相同记录类型的计数。

[0013] 所述的单独的监控数据处理线程的数据输出工作为屏幕打印或进行磁盘文件写入或通过网络传输数据至远程的数据库服务器。

[0014] 所述的流控检测周期设为 0，表示始终不执行流控；如果数据输出请求数阈值设为 0，表示始终执行流控机制。

[0015] 所述的监控数据异步采集处理方法需在采集模块初始化时选定异步数据输出请求队列长度，队列长度过小会导致大量监控数据无法入队而被丢弃，所述的异步数据输出请求队列长度选定方法如下：

[0016] 监控事件数据的到达和监控事件数据的落地之差即为该时间段的队列长度的变

化量 Δ，在上一个交易日 D_{T-1}，令 e_i 表示第 i 秒内发生的事件数，对每条采集的数据定义了 m 个输出目标，每个输出目标可以通过性能测试数据的平均值获得其单条数据处理时间的期望值 $\overline{\mu}$，那么单位时间内 m 个目标的总服务速度（可以处理的数据）为 $v_m = 1/\sum_{i=1}^{m}\overline{\mu}_i$，综上，第 i 秒对队列的改变量为：

[0017] $\Delta = e_i - v_m$ (1)

[0018] 设第 i-1 秒后的队列长度为 L_{i-1}，那么第 i 秒后的队列长度即为：

[0019] $L_i = \max(L_{i-1} + \Delta, 0)$ (2)

[0020] 注意因为队列长度不可以为负数，所以在服务速度大于数据到达速度的场景下，队列最小为 0；

[0021] 交易时段总长为 t，那么在 D_{T-1} 的整个交易时段内队列长度出现过的最大值便可以作为未来队列长度的基准值 L_q：

[0022]

$$L_q = \left\lceil \alpha\left(\max_{i\in[1,t]}(L_i)\right)\right\rceil + c, (\alpha \geq 1, c > 0) \qquad (3)$$

[0023] 其中 α 为调和系数，即对历史数据中最坏情况下的队列长度再进行一定系数的放大，以保证未来使用该基准值后，监控数据被丢弃的概率很小，c 为某个正整数常量，用于保证队列基准值的最小值，其选取值可以是任意一秒的非零事件数 e_i（$e_i > 0$）。

[0024] 所述的流量控制具体方法如下：

[0025] 将一个流控周期 T_w 划分为流控检测阶段 T_d 和流控执行阶段 T_e，定义 $\lambda = m/T_d$，其中，λ 为数据到达速度的上限阈值，m 为数据输出请求数阈值，数据到达速度超过该阈值时，认为磁盘读写会对系统性能产生不良影响，对于相同的 λ 值，m 设置的越低，T_d 就越低，即在一个周期内会越早进入流控执行阶段，系统总体性能的损耗会越少；另一方面，越早进入流控执行阶段，就有越多的监控信息被压缩合并，对输出结果的质量产生削弱。

[0026] 本发明同现有技术相比，其优点在于：通过流速权控制算法，保证关键实时系统的平稳运行，能够有效地满足不同监控目标的监控数据采集需求，在突发大量落地请求时，IO 数据读写的性能损耗不会超过给定的阈值，并通过特征数据的合并方法，保证了落地数据的质量；对算法涉及的关键参数进行了理论分析，给出了计算方法，从而指导用户使用，适用于证券交易系统等大型关键实时系统的状态监控和日志记录场景。

[附图说明]

[0027] 图 1 是本发明监控数据异步采集处理结构图；

[0028] 图 2 是本发明流控周期划分示意图；

[0029] 图 3 是本发明中流控合并机制示意图；

[0030] 指定图 1 作为本发明的摘要附图。

[具体实施方式]

[0031] 下面结合附图对本发明作进一步说明，这种系统的结构和原理对本专业的人来说

是非常清楚的。应当理解，此处所描述的具体实施例仅仅用以解释本发明，并不用于限定本发明。

[0032] 实施例 1

[0033] 图 1 是本发明的监控数据异步采集处理结构图，本部分设计的工作原理是通过调用 Linux 系统线程接口，监控模块在初始化后启动一个单独的监控数据处理线程（ProcessThread）负责系统数据的多目标处理输出，主程序每次调用数据采集接口时，实际上在内部仅仅将系统状态数据缓存到一个内存中的数据输出请求队列，并没有产生实际的 IO 操作；那么采集接口在完成入队列操作后，会立即返回并让主程序继续执行其实时应用逻辑，从而在本质上避免了同步操作可能产生的 IO 吊住主程序的危险。

[0034] 监控数据处理线程（ProcessThread）不停轮询数据输出请求队列，一旦发现有新的请求时，就从队头弹出数据记录并执行真正的输出请求（可能为多目标输出，如同时打印到屏幕和写入文件），一直到将队列中的记录处理完毕。

[0035] 监控数据处理线程（ProcessThread）的数据输出工作可能是简单的屏幕打印，可能是进行磁盘文件写入，也可能通过网络传输数据至远程的数据库服务器，但是因为有独立线程执行该操作，所以无论其性能或延迟消耗有多大，都不会阻塞主程序业务逻辑的迅速执行和快速响应。

[0036] 实施例 2

[0037] 监控数据采集库的采集接口在进行数据入队操作时，会实时检查系统监控事件的发生频率，并动态调整采集数据的流量，从而避免大量数据 IO 操作对系统整体性能的冲击。该部分设计如下：系统设定 3 个参数，分别是流控检测周期（rateControlInterval），数据输出请求数阈值（rateControlCount）和流控执行周期（rateControlDuration）。流控算法首先在流控检测周期内计数到达队列的数据请求数目，一旦该请求数达到或超过请求数阈值，就立即开启流量控制机制，并一直持续到流控执行周期的时间段结束。因此，动态流控的一个总周期可以划分为两个阶段：流控检测阶段 T_d 和流控执行阶段 T_e，如图 2 所示，其中 m 为检测阶段的数据输出请求数阈值。此外，如果流控检测周期设为 0，表示始终不执行流控；如果数据输出请求数阈值设为 0，表示始终执行流控机制。

[0038] 实施例 3

[0039] 在流控执行周期，监控数据采集库的采集接口不再将新到的数据加入数据输出请求队列，而是通过初始化时预先建立好的哈希表进行相同特征数据的合并，从而实现降低实际数据流量的目标。需要根据自己的业务对“相同特征数据”（即 Hash Key）的概念进行定义，例如对于证券交易系统，如果监控数据来自相同的 PBU 或者产品号，那么可以定义为相同的特征数据。数据的合并机制是：对于未出现过的特征数据类型，在 e 哈希表里面增加一条；否则，只增加相同记录类型的计数。

[0040] 以系统错误的监控为例：在大量错误突发时，采集接口以“错误类型号”为 Hash Key 相同特征数据写入每条错误数据，之后对于相同类型的错误只增加错误的发生次数。流控执行周期结束后，采集模块会将哈希表里面的记录统一写入数据输出请求队列，等待 ProcessThread（监控数据处理线程）进行处理，同时开启下一轮的流控检测周期。因为对于相同的特征数据进行了合并，所以每个周期需要实际分发和落地的数据量大大降低了。

[0041] 图 3 的是上述流控机制的具体应用例：其中，流控检测周期设为 3 秒，数据输出请

求数阈值设为100，流控执行周期为7秒。可以看到，在第3秒的第101个错误发生时，流控机制开启；此后，对于重复出现的错误类型只记录重复次数，并在流控结束后输出哈希表中的内容，包括(错误号，次数)。

[0042] 实施例4

[0043] 流量动态可控的监控数据异步采集算法通过监控数据异步采集处理的设计，将数据落地逻辑和主应用程序逻辑解耦，避免了磁盘IO堵住主线程的危险；通过流量控制开启检测算法，实时检测系统状态，动态开启流控机制；通过流控数据合并机制，既有效地降低了实际落地的数据量，又记录了所有监控的特征数据类型，保证了采集的数据质量。

[0044] 为优化本算法的实施效果，以下对算法的两部分关键参数：数据输出请求队列长度和流控周期参数的选取策略进行介绍。

[0045] 异步数据输出请求队列长度选取策略：

[0046] 数据输出请求队列是主线程和监控数据处理线程之间缓存数据的桥梁，异步采集机制要求在采集模块初始化时logQueueLen（输入队列长度参数），如果队列长度过小会导致大量监控数据无法入队而被丢弃。虽然内存空间是个比较富裕的资源，但是仍然需要基于其自身系统的应用场景和性能需求计算队列长度基准值。

[0047] 从排队论的角度看：监控事件数据的到达相当于顾客，而数据的落地相当于服，在单位时间段中，“到达顾客数”与“服务完成数”之差即为该时间段的队列长度的变化量Δ。假设历史数据可以反映事件到达率的分布，那么对于需要监控和采集的事件，首先统计其历史数据：以交易业务为例，在上一个交易日D_{T-1}，令e_i表示第i秒内发生的事件数；如果期望对每条采集的数据定义了m个输出目标，每个输出目标可以通过性能测试数据的平均值获得其单条数据处理时间的期望值$\overline{\mu}$，那么单位时间内m个目标的总服务速度(可以处理的数据)为$v_m = \Big/ \sum_{i=1}^{m} \overline{\mu}_i$。综上，第i秒对队列的改变量为：

[0048] $\Delta = e_i - v_m$ (1)

[0049] 设第i-1秒后的队列长度为L_{i-1}，那么第i秒后的队列长度即为：

[0050] $L_i = \max(L_{i-1} + \Delta, 0)$ (2)

[0051] 注意因为队列长度不可以为负数，所以在服务速度大于数据到达速度的场景下，队列最小为0。

[0052] 假设交易时段总长为t，那么在D_{T-1}的整个交易时段内队列长度出现过的最大值便可以作为未来队列长度的基准值L_q：

[0053]

$$L_q = \left\lceil \alpha\left(\max_{i\in[1,t]}(L_i)\right)\right\rceil + c, (\alpha \geq 1, c > 0) \qquad (3)$$

[0054] 其中α为调和系数，即对历史数据中最坏情况下的队列长度再进行一定系数的放大，以保证未来使用该基准值后，监控数据被丢弃的概率很小，c为某个正整数常量，用于保证队列基准值的最小值，其选取值可以是任意一秒的非零事件数e_i（$e_i > 0$）。

[0055] 例如，假设D_{T-1}的交易时间为5秒，系统每秒平均处理2条数据，α选取1.8，c选取1那么对于如下事件发生率e_i，可以采用上述公式计算队长基准值：

[0056] 即$L_q=\lceil 1.8\times 3\rceil+1=7$。

[0057] 为了让计算结果更代表一般情况，可以对 n 天的历史数据 D_{T-1}～D_{T-n} 分别进行统计并计算 L_q，再最终选取其中的最大值作为异步队列长度基准值。

[0058] 流控参数选择策略：

[0059] 一个流控周期 T_w 划分为两个阶段：流控检测阶段 T_d 和流控执行阶段 T_e，如图 2 所示，定义 $\lambda = m/T_d$（m 为数据输出请求数阈值）为数据到达速度的上限阈值，即一旦超过该阈值，便认为磁盘读写会对系统性能产生影响，可基于自己的系统参数和业务需求选定该值。但是，对于相同的 λ 值，m 设的越低，T_d 就越低，即在一个周期内会越早进入流控阶段，系统总体性能的损耗会越少；但是另一方面，越早进入流控阶段，就有越多的监控信息被压缩合并，对输出结果的质量产生削弱。

[0060] 将流控参数 m 的选择问题定义如下：

[0061] a. 在一个周期内，数据采集对系统性能的影响必须低于某个阈值；

[0062] b. 在一个周期内，流控机制对数据质量的损耗必须低于某个阈值；

[0063] c. 在满足 a 和 b 的前提下，最小化流控机制对性能的影响和数据质量的影响。

[0064] 因为$T_d=\frac{m}{\lambda}$以及 $T_e=T_w-T_d$，所以对于给定的一个流控周期，一旦确定数据输出请求数阈值 m，便可以计算出其他流控周期长度，所以计算出 m 是流控参数选择的关键。

[0065] 为解决该问题，需要对性能花费和数据质量这个两个概念进行定义。从总体看，性能花费正比于一个周期内需要输出的数据记录数，而数据质量的损耗正比于该周期内合并的记录数，因此可以用一个周期中 IO 输出的监控数据量分别描述这两个指标。综上，令 λ 表示可以接受数据到达速度的上限，令 h 为使用哈希表进行数据合并的压缩率，令 T_w 为一个流控周期，那么该时间段的性能影响 y_p 和数据质量损耗 y_q 分别为：

[0066] $$y_p=m+\left(T_w-\frac{m}{\lambda}\right)\lambda(1-h) \qquad (4)$$

[0067] $$y_q=\left(T_w-\frac{m}{\lambda}\right)\lambda h \qquad (5)$$

[0068] y_p 是流控检测阶段和流控执行阶段需要处理的数据之和，而 y_q 是流控执行阶段合并的数据量，因此，流控参数 m 的选择问题可以定义为一个一维线性规划问题：

[0069] $\min y = (\omega_1 y_p+\omega_2 y_q)$

[0070] s.t.

[0071] $y_p \leqslant c_1 \lambda T_w, y_q \leqslant c_2 h \lambda T_w, m \in [0, \lambda T_w]$ (6)

[0072] $\omega_1+\omega_2=1$

[0073] $\omega_1, \omega_2, c_1, c_2 \in [0,1]$

[0074] 其中线性规划目标函数 y 为性能花费和质量损耗的加权平均值，目标求解最小化该值的流控参数 m（即问题定义的 c），$y_p \leqslant c_1 \lambda T_w$ 和 $y_q \leqslant c_2 h \lambda T_w$ 分别描述性能花费必须低于某个阈值，而数据质量损耗也必须低于某个阈值(即问题定义的 a 和 b)。

[0075] 对于一维线性规划，通过函数作图法可以方便地求出最优解；此外，也可以使用 Mathematica 等数学软件求出最优解，求出 m 后，就可以得出$T_d=\frac{m}{\lambda}$以及 $T_e=T_w-T_d$。

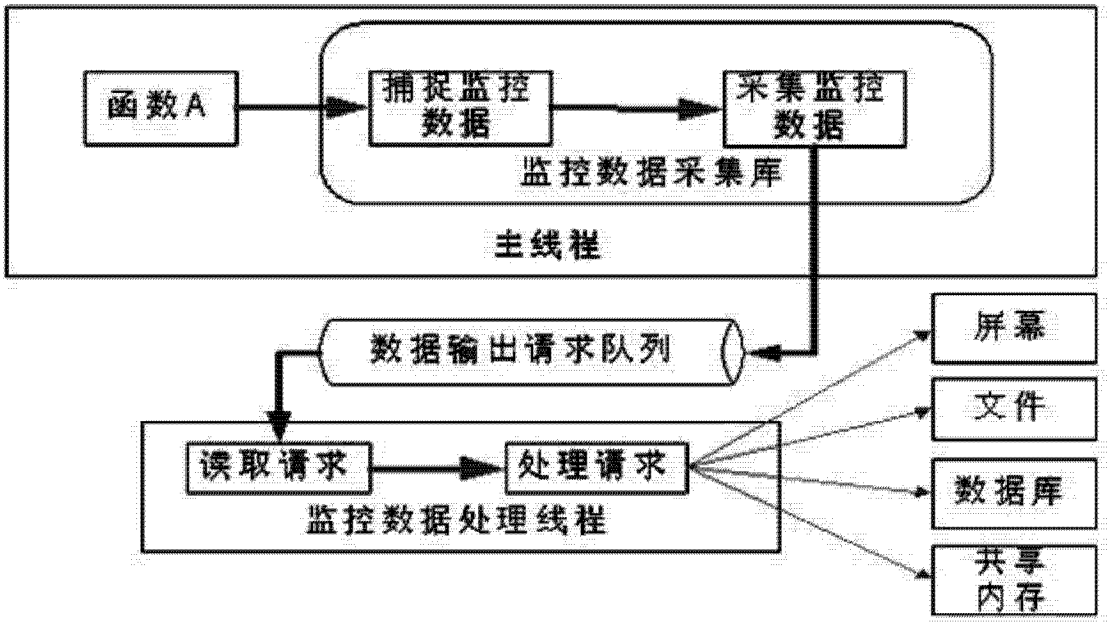

函数A
捕捉监控数据
采集监控数据
监控数据采集库
主线程
数据输出请求队列
屏幕
文件
数据库
共享内存
读取请求
处理请求
监控数据处理线程

图 1

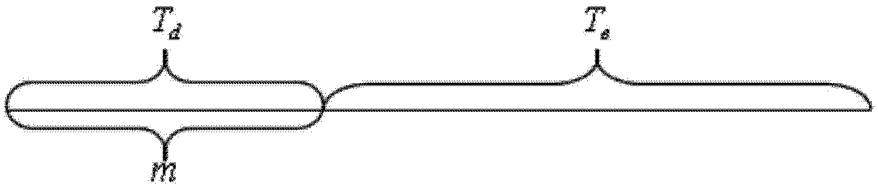

T_d
T_e
m

图 2

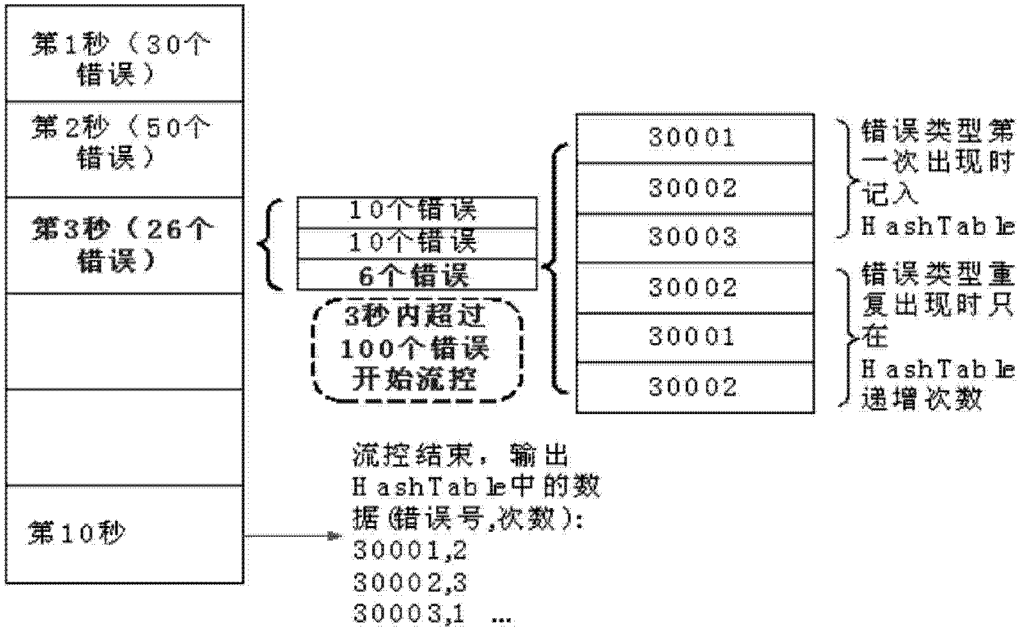

第1秒（30个错误）
第2秒（50个错误）
第3秒（26个错误）
第10秒
10个错误
10个错误
6个错误
3秒内超过100个错误开始流控
30001
30002
30003
30002
30001
30002
错误类型第一次出现时记入HashTable
错误类型重复出现时只在HashTable递增次数
流控结束，输出HashTable中的数据(错误号,次数):
30001,2
30002,3
30003,1 ...

图 3