REPS:一种高效的容错并行概率流Skyline查询方法

张卫华 李小勇 马 俊 余 杰
(国防科技大学计算机学院 长沙 410073)

摘 要 概率数据流的并行Skyline查询作为当前大数据分析的一个重要方面,在诸多实际应用中发挥着重要作用。针对并行概率流Skyline查询过程中因发生故障而导致查询结果不准确和查询中断等问题,提出了一种基于复制的容错并行Skyline查询方法REPS。该方法选择参与并行处理的计算节点作为副本节点,并采用层次-循环式数据副本放置策略,选择优先级高的副本恢复数据来保证数据恢复的高效性;同时将故障检测、丢失数据恢复和查询过程恢复贯穿于整个查询更新过程中,以减少容错处理的额外通信和计算开销,并实现快速的容错并行查询。实验结果表明,REPS方法不仅在无故障发生和单个节点失效时具有较高的查询处理效率,而且对于多节点失效情形,仍然能够保持较高的查询处理速率且满足查询需求。

关键词 概率Skyline,容错查询,数据流,并行查询,大数据
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.8.046

REPS:An Efficient Fault-tolerant Approach for Parallel Skyline Queries over Probabilistic Data Streams

ZHANG Wei-hua LI Xiao-yong MA Jun YU Jie
(College of Computer,National University of Defense Technology,Changsha 410073,China)

Abstract The parallel Skyline query over probabilistic data streams,as an important aspect of big data analysis,plays an important role in various applications. To deal with the problem that the query results may be incorrect and interrupted,due to various faults occurred in the process of parallel Skyline queries over probabilistic streams,a replication-based fault-tolerant distributed parallel Skyline query scheme named REPS was proposed. Specifically,the compute nodes are also regarded as the replication nodes,and a layer-alternation strategy for replica placement is proposed in REPS. Thus,the lost data can be recovered efficiently by selecting the replicas with high priority. Moreover,the processes of fault detection,data recovery and query recovery are throughout the whole query updating process,in order to reduce the communication and computation overhead and achieve rapid fault-tolerant parallel query processing. Extensive experimental results demonstrate that REPS method not only has high efficiency when no failure occurs or a single node fails,but also can maintain a high processing rate and meet the query requirement even when multiple failures occur.

Keywords Probabilistic Skyline,Fault-tolerant queries,Data streams,Parallel queries,Big data

不确定数据流作为一种典型的大数据类型,对其进行查询分析已成为大数据分析的一个重要方面[1]。不确定数据流的Skyline查询在信息检索、数据挖掘、决策制定和环境监控等众多应用中发挥着重要作用[2],目前已成为数据库领域的一个研究热点。近年来,随着数据流应用的广泛兴起和发展,如何高效处理不确定数据流的Skyline查询成为当前亟待解决的问题[3]。尽管目前已有不少针对数据流Skyline查询的方法被提出,例如针对确定性数据流Skyline查询的研究[4-12]和针对不确定数据流Skyline查询的研究[13-16]。然而,前者由于数据不确定性的引入,使得其难以适用于不确定数据流的Skyline查询[17];而后者主要采用集中式的处理方式,由于查询需求的不断扩大及其在计算过程和数据流更新方面的复杂性,导致集中式方法难以满足现实应用对查询效率的需求。当前诸如云计算数据中心、网格计算、多处理器集群和高速局域网等高带宽的分布式计算环境的兴起和广泛运用,为实现并行查询处理提供了有利条件。为此,在前期工作中,研究小组针对并行不确定数据流Skyline查询问题展开了深入的研究,并提出了多种并行查询模型和优化算法[18]。

尽管高带宽的分布式计算环境为不确定数据流的并行Skyline查询处理提供了良好条件,但是也带来了一系列的挑战,其中最为典型的是容错问题。例如,云计算数据中心环境往往规模巨大,通常由成千上万的节点以集群形式构成分布式系统。尽管现代数据中心采用的服务器及各种软硬件资源已具备较好的性能,但是在大规模的数据中心内部,依然存在各种会导致系统发生故障的因素。据统计,在Google运行的MapReduce应用中,平均5台工作机便有一台会发生失

到稿日期:2014-09-28 返修日期:2015-01-03 本文受国家自然科学基金项目(61303191,60873215),国家重点基础研究发展规划(973)项目(2011CB302601)资助。

张卫华(1977—),男,博士,副研究员,主要研究领域为操作系统、云计算、数据库查询等;李小勇(1982—),男,博士,助理研究员,主要研究领域为数据流管理、网络计算、数值并行计算等,E-mail:sayingxmu@163.com;马 俊(1983—),男,博士,助理研究员,主要研究领域为操作系统、信息安全防护等;余 杰(1982—),男,博士,助理研究员,主要研究领域为操作系统、云计算等。

效[19]，且在4000台计算节点的集群上运行6个小时MapReduce任务的过程中，最少有一个节点会发生磁盘失效故障[20]。由此可见，数据中心环境中故障的发生应被视为常态而非例外情形来对待。若在并行查询的过程中发生节点或者网络中断等故障，将引起查询结果出错、查询中断且浪费大量的系统资源等问题，严重影响着用户的查询体验。在查询过程中加入容错机制，以降低故障对查询处理过程和结果的影响，是当前查询处理研究必须考虑的问题。

在并行Skyline查询研究领域中，尽管目前针对容错Skyline查询方面的研究很少，但是不少研究已经意识到了容错查询的重要性，并试图通过各种方法来改进和解决容错查询问题。例如，Wu等[21]采用CAN结构组织网络来避免使用集中节点而引起单点失效等情况发生；Wang等[22]将搜索空间进行自适应划分来减轻查询过程中出现的“热点”问题，并且通过静态负载划分和动态负载迁移来实现负载均衡。尽管以上各种优化策略能够有效提高查询的容错性能，却无具体的解决故障发生后的容错查询处理策略。此外，王媛等[23]提出了一种基于多副本的容错分布并行查询算法，以解决确定性数据集上的容错并行Skyline查询问题。然而，该方法主要针对确定性数据集上的查询问题，而无法运用于数据流和不确定数据的容错并行查询中。因此，为提高并行概率流Skyline查询的容错能力，提出了一种基于复制的容错并行Skyline查询方法REPS（REplication based fault-tolerant Parallel Skylining），该方法不仅能够有效解决容错并行查询问题，而且能够达到较高的性能水平。

1 基本概念与问题描述

1.1 相关概念

概率数据流（简称概率流），是一种特殊的不确定数据流类型，其中，构成数据流的单个流元组为概率型不确定数据。通常描述概率型不确定数据的概率模型包括点概率模型、概率分布模型和概率密度函数[2,24]3种。本文研究的概率流元组是指点概率模型表示的流元组，例如$\langle(a_1,a_2,\cdots,a_n),p\rangle$，其中$(a_1,a_2,\cdots,a_n)$表示确定的元组信息，而$p$则表示该元组的存在概率，该不确定数据流类型也是当前不确定数据流Skyline查询研究的主要对象和未来趋势之一[14,18,25,26]。此外，本文研究的概率流Skyline查询主要关注于基于计数的滑动窗口上的连续Skyline查询，且主要针对增量（Append-Only）数据流模型，即在流元组过期前不存在元组被删除或者修改的现象。不确定数据流中的元组按照先到先服务的方式进行处理，且最先到达的流元组最先过期。此外，为便于描述，采用整数值并依据流元组的到达顺序标记各流元组的位置。假定流元组e在数据流中的到达次序采用整数$k(e)$标记，即代表e为数据流中第$k(e)$个到达的元组，且将整个概率流表示为DS。由于采用基于计数的滑动窗口模型来建模分析该概率流，因此查询时将始终关注DS中最新到达的滑动窗口中的元组。特别地，假定W表示滑动窗口中所有元组的集合，且$|W|$表示该集合中元组数目。此外，不失一般性，本文中假定对于所有概率流中元组的确定性属性值均以小为优。

定义1（流元组支配） 对于概率数据流中的任意两个具有n个确定性维度的流元组a和b，称a支配b（记为$a\prec b$），当且仅当在所有维度$1\leqslant i\leqslant n$上，均满足$a.i\leqslant b.i$，且至少存在某一维度j，使得其满足$a.j<b.j$。

需要特别指出的是，上述定义中所指出的n个维度，是指不确定流元组的确定性属性维度，而并不包含流元组的存在概率维度。流元组的支配关系，与其存在概率的大小并无任何关系。在上述流元组支配定义的基础上，可得出对于W中的任意元组e，其成为Skyline元组的概率$P_{sky}(e)$为[14,17]：

$$P_{sky}(e)=P(e)\times\prod_{e'\in W,e'\prec e}(1-P(e')) \tag{1}$$

在此基础上，可进一步给出对于某一时刻滑动窗口W中概率阈值Skyline（即q-Skyline）集合（简记为$SKY_{W,q}$，其中q表示概率阈值）的概念如下：

定义2（q-Skyline集合） 给定概率阈值q，滑动窗口W中的q-Skyline集合定义为W的一个子集，其中每个流元组成为W中Skyline元组的概率均不小于概率阈值q，即$\forall e\in SKY_{W,q}$，均满足条件$P_{sky}(e)\geqslant q$。

此外，为了便于本文后续论述，根据$P_{sky}(e)$公式中支配e的对象在到达时间上早于或晚于e，可分别定义公式$P_{new}(e)=\prod_{e'\in DS_N,e'\prec e,k(e')>k(e)}(1-P(e'))$和公式$P_{old}(e)=\prod_{e'\in DS_N,e'\prec e,k(e')<k(e)}(1-P(e'))$，则式(1)可采用另一种形式来表示，即$P_{sky}(e)=P(e)\times P_{new}(e)\times P_{old}(e)$[14]。其中，$P_{new}(e)$表示那些比$e$更早到达且支配$e$的所有元组均不存在的概率，而$P_{old}(e)$表示那些比$e$更晚到达且支配$e$的所有元组均不存在的概率。在此基础上，可进一步定义W中候选q-Skyline集合$C_{q,W}$的概念如下：

定义3（$C_{q,W}$集合） 对于任意元组$e\in W$，$C_{q,W}$表示W中部分元组的集合，且该集合中的任意元组e均满足$P(e)\times P_{new}(e)$的值不小于概率阈值q，即$S_{N,q}=\{e\mid e\in DS_N\wedge P(e)\times P_{new}(e)\geqslant q\}$。

由定义3可知，对于W中的某个流元组e，若其不属于$C_{q,W}$，则e必然不为W中的q-Skyline元组。因此，在对W中的元组进行Skyline查询时，只需关注$C_{q,W}$中的元组即可。

1.2 问题描述

本文主要针对集中到达的概率流，利用分布式计算环境（如云计算数据中心）进行容错并行Skyline查询处理。与已有研究[14-16]类似，本文主要关注于即时的连续查询处理，即到达一个新的流元组立即进行一次查询处理更新。本研究所采用的分布式网络环境如图1所示，该结构也是目前数据流并行查询处理典型的结构之一，本研究小组在前期的工作中同样采用了该分布式结构[18]。

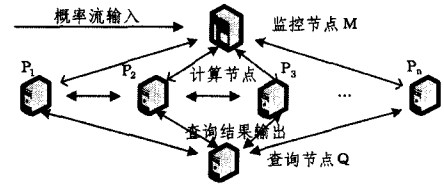


图1 分布式网络结构

结构中主要包括3类节点，即监控节点M、计算节点P_i（$1\leqslant i\leqslant n$）和查询节点Q[18]。其中，监控节点主要负责接收新到达的流元组，并且将其分发至各计算节点，以及协调节点之间的查询计算等；计算节点主要负责参与与执行计算相关的任务，是并行查询处理的主体；查询节点则负责收集各计算节点返回的查询结果，并将合并的最终q-Skyline结果呈现给查询的用户。

在分布式计算环境中，通常故障发生的情形和类型较多，如节点故障、通信故障和软件故障等。即便如此，本文将各种

故障统一称为节点故障，其主要原因在于：在查询处理过程中，计算节点之间通过相互发送探测消息确定其他节点的状态；若某个计算节点在一定的时间内未返回应答消息，则该计算节点视为失效；尽管该计算节点可能因为通信等原因而未能及时应答，但由于该计算节点无法参与并行查询处理而仍将其视为失效。在上述数据流查询模型和分布式网络结构的基础上，可将本文研究的主要问题描述如下。

问题描述：对于一组高速且连续到达的概率流 DS，采用基于计数的滑动窗口对其进行建模，研究一种基于高带宽的分布式计算环境（如数据中心环境），连续即时地查询 W 中的 q-Skyline 元组集合的并行查询方法，该方法即使在单个甚至多个计算节点失效的情形下，仍然能够快速恢复查询过程并及时地返回查询结果。

2 并行查询处理

为实现并行处理，需要有效地分配查询处理任务，并协调各计算节点完成整个查询任务。由于查询主要针对于全局滑动窗口上的流元组进行，因此可将全局滑动窗口按照某种方式划分为多个局部滑动窗口，并将各个局部窗口上的流元组映射至各计算节点，从而实现多个计算节点的并行查询处理。在划分的过程中，采用完全不重叠的划分方式，使得各计算节点能够独立地针对各自局部窗口中的元组展开计算。

为了实现数据流的并行查询处理，在设计并行查询处理框架时，同样采用了前期工作[18]中所采用的全局滑动窗口划分的方式。假定查询系统中存在 n 个计算节点 $P_i(1\leqslant i\leqslant n)$，则将其在逻辑上划分为 n 个局部滑动窗口 $W_i(1\leqslant i\leqslant n)$，且该划分满足完全不重叠划分，即满足条件 $W=\bigcup_{i=1}^{n}W_i$ 和 $W_i\cap W_j=\varnothing(i\neq j)$。在并行处理过程中，各局部滑动窗口上的处理任务分别由对应的计算节点来完成。特别地，各计算节点分别负责其局部滑动窗口内元组的 Skyline 概率更新，同时负责对新到达的流元组支配概率的计算；对于新到达的流元组所映射的计算节点，其还需负责计算新到达元组的全局 Skyline 概率。

假定监控节点 M 接收到新的流元组 e_{new}，则 M 首先将其发送至全部计算节点，并将其映射至某个计算节点（如 P_i），即 e_{new} 所属的计算节点。所属关系表明在后续处理过程中，将由 P_i 负责计算 e_{new} 的全局 Skyline 概率。由于所有的计算节点均需要根据新到达的流元组更新各自局部窗口内元组的 Skyline 概率，因此 M 将 e_{new} 直接发送至所有计算节点，以避免通过其它计算节点再次转发。在查询处理过程中，过期的计算节点需要将过期的元组 e_{old} 发送至其它节点，使得各计算节点能够同时根据 e_{old} 和 e_{new} 来更新其局部窗口中元组的 Skyline 概率。当各计算节点 $P_j(1\leqslant j\leqslant n, j\neq i)$ 接收到最新到达的流元组 e_{new} 后，即可根据公式 $P_d^j(e_{new})=\prod_{e_i\in W_j, e_i\prec e_{new}}(1-P(e_i))$ 直接计算出 e_{new} 相对于该局部滑动窗口 W_i 的支配概率值 $P_d^j(e_{new})$。当各计算节点计算出 $P_d^j(e_{new})$ 值之后，分别将其发送至 e_{new} 所映射的计算节点 P_i。因此，当计算节点 P_i 获得所有计算节点返回的支配概率之后，即可根据公式 $P_{sky}(e_{new})=P(e_{new})\prod_{j=1}^{n}P_d^j(e_{new})$ 快速计算出 e_{new} 的全局 Skyline 概率值。同时，各计算节点均根据所接收到的 e_{new} 和 e_{old} 元组信息，采用如下公式更新其各自局部滑动窗口中元组的 Skyline 概率：

$$P_{sky}(e)=\begin{cases}P_{sky}(e)*(1-P(e_{new})), & \text{if } e_{new}\prec e\\ P_{sky}(e)/(1-P(e_{old})), & \text{if } e_{old}\prec e\end{cases}\quad(2)$$

当所有更新操作完成后，各计算节点将满足条件 $P_{sky}(e)\geqslant q$ 的所有元组发送至查询节点 Q。需要注意的是，各计算节点对于其所维护的局部滑动窗口中的元组，并非需要对所有的元组进行概率更新计算，而是仅需对有希望成为最终 q-Skyline（即属于 $C_{q,w}$ 集合）的元组进行概率更新计算。因为在各局部窗口中，不满足以上条件的元组最终不可能成为全局 q-Skyline 元组。

3 容错并行查询

3.1 容错策略设计

数据复制作为分布式系统中的一种重要技术，是实现数据容错的关键技术之一，目前在云计算数据中心环境中运用广泛。由于本文研究的数据流查询需要及时响应和快速查询处理，且所有处理的流数据均维护在内存中，相对于编码策略的数据容错技术，复制策略更适合于数据流的容错查询处理。因此，在 REPS 方法中选择复制策略来实现数据容错。

在设计容错策略时，主要需要考虑解决 4 个方面的问题：(1)确定各副本中复制的内容，以保证查询结果的正确性，同时又尽可能地减少查询所需的通信开销和内存开销；(2)确定触发数据复制和副本更新的时机或事件，以满足用户减少查询等待时间和保证副本一致性的要求；(3)确定副本的数目，以保证即使在单个节点甚至多个节点发生故障时，仍然能够保证数据的可用性；(4)确定数据副本的放置策略，以实现高效的并行查询处理和减少参与并行处理的计算节点的数目。

首先，在各副本中需要复制的数据内容取决于查询过程中的计算需求。对于任意的计算节点 P_i，其维护着滑动局部窗口 W_i，则在查询过程中需要复制 W_i 中的所有元组，并记录其中所有候选 Skyline 元组集合中 C_{q,W_i} 元组的 Skyline 概率值。由于只有在 C_{q,W_i} 中的元组才可能成为最终的 q-Skyline 元组，因此在查询处理过程中无需计算和更新那些不在集合 C_{q,W_i} 中的元组的 Skyline 概率。其次，数据复制的时机与并行查询算法密切相关。在 REPS 方法中，采用周期性更新数据副本的方式，且副本更新周期 $\mathcal{T}$ 与查询处理的需求和计算节点的失效率紧密相关。假定在 M 处记录流元组总共的更新次数为 $\mathcal{X}$，若 $\mathcal{X}\ \%\mathcal{T}$ 的值等于 0，则 M 将通知所有的计算节点更新其相应的副本信息。再次，副本数目的选择主要取决于计算节点的失效率以及同时失效的节点数目。在设计处理算法时一般认为 2 个或更多个计算节点同时发生失效的概率极低[27]，所以主要考虑选择 2 个副本数目来实现容错处理。实际上，所提出的 REPS 方法能够同时应对多个计算节点同时失效，在后续论述中将会对此展开阐述。最后，为减少参与并行计算的计算节点数目，在处理过程中选择参与计算的计算节点作为存储副本的节点。因此，在每个计算节点上实际维护了多个局部滑动窗口信息，包括一个主滑动窗口（即本节点对应的局部滑动窗口）和多个其它计算节点的局部滑动窗口的副本信息，具体的副本放置策略将在后面论述。

3.2 数据副本放置

在研究快速数据恢复算法的过程中，首先需要解决数据副本的放置问题。研究副本放置的目的在于尽量平衡各计算

节点上的负载，同时提高副本数据的整体可用性，避免当节点发生失效时，多个节点从同一个节点获取数据而导致整个数据恢复的效率低下。

假定整个系统中有 n 个计算节点参与并行计算 $P_1, P_2, \cdots, P_n$，且在每个计算节点上均维护了用于恢复丢失数据的 k ($k<n$)个其它计算节点的数据副本，通常 k 不超过 3。为了实现快速的数据恢复，REPS 采用了一种层次-循环式的数据副本放置策略。该策略可简要描述为：首先，对于某个计算节点 P_i，将其所维护的多个窗口进行层次性地区分，例如第一层副本记为 R_i^1，第二层副本记为 R_i^2，而第 l 层副本记为 R_i^l，且各层副本的优先级依次递减；同时，为了避免计算节点上的两个或者更多个数据副本放置于同一个计算节点上，采用循环式的副本放置规则，该规则可形式化地描述为：

$$R_i^l=\begin{cases}(P_i-l), & \text{if } P_i>l\\ n+P_i-l, & \text{if } P_i\leqslant l\end{cases} \tag{3}$$

式中，P_i 表示计算节点的编号，且满足 $1\leqslant l\leqslant k$。为了更清楚地阐述上述数据副本的放置规则，以下将选择 $k=2$ 作为实例进行阐述。为便于论述，假定每个计算节点维护着 3 个副本窗口 P、S 和 T，且三者的优先级满足 $P>S>T$。若计算节点 $P_1, P_2, \cdots, P_n$ 上维护的主滑动窗口分别为 $1-P, 2-P, \cdots, n-P$，则根据式(3)可知，这些计算节点上对应的次级窗口分别为 $n-S, 1-S, \cdots, (n-1)-S$，而第三级窗口分别为 $(n-1)-T, n-T, \cdots, (n-2)-T$。假定 $n=10$ 且 $k=2$，则其数据副本的具体放置情况如图 2 所示。

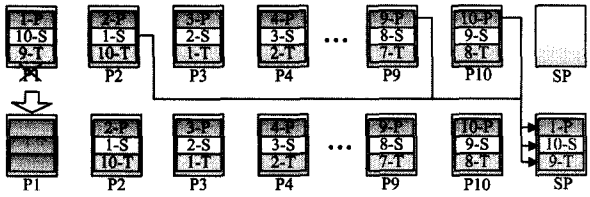


图 2　数据放置及单个计算节点失效时的数据恢复示例

为了避免从同一个计算节点上获取丢失的数据而引起通信阻塞或者更多的通信传输延迟，可按照计算节点副本优先级的方式来获取数据副本信息。在选择具体的副本节点恢复丢失的数据时，总是选择优先级最高的副本节点。如图 2 所示的实例中，若计算节点 P_1 失效，则由计算节点 SP 取代 P_1，且从 P_2 处获取 $1-S$ 副本信息以恢复丢失的 $1-P$ 信息，而从 P_{10} 和 P_9 处分别获取 $10-S$ 和 $9-T$ 信息。同样的数据放置规则和副本数据获取方式可用于多个计算节点失效的情形。

3.3　快速数据恢复

根据 3.1 节中的讨论可知，每个计算节点周期性地更新其存储于其它计算节点上的所有副本信息。该副本信息包括其局部滑动窗口中维护的所有元组信息及其候选元组的 Skyline 概率信息。特别地，在 M 中维护了两个优先队列 NQ 和 OQ，分别用于记录最新元组和过期元组的相关信息。此外，M 处还维护了一个计算节点列表以记录所有计算节点的相关配置信息。各计算节点的编号由 M 统一按序分配，且该排序和编号在查询处理过程中一直不变。当某个计算节点失效时，接替失效节点的新节点仍然采用此失效节点的编号。假定 PR_f 表示含有计算节点 P_f 副本的所有计算节点的集合，则当某个计算节点 P_f 失效时，可采用如下算法恢复由于该节点失效而丢失的相关数据信息。

算法 1　快速数据恢复算法

输入：P_f：失效的计算节点；P_s：将替代 P_f 的空闲节点；
　　NQ：最新元组优先队列；OQ：最近过期元组优先队列
输出：在 P_s 中恢复 P_f 所丢失的元组信息
1. M 选择空闲节点 P_s 替代 P_f，并告知 P_s 其它计算节点信息；
2. M 查找节点列表并获取 P_s 对应的副本节点集合 PR_f；
3. M 发送 NQ、OQ 和 PR_f 信息至计算节点 P_s；
4. P_s 从 PR_f 中的计算节点处获取丢失的副本信息；
5. P_s 利用 NQ 和 OQ 根据式(2)恢复所有丢失的计算结果；
6. P_s 通知 M 所有丢失的数据已恢复；

此外，假定流元组总共更新的次数为 $\mathcal{X}$，而副本更新的周期为 $\mathcal{T}$，则当 $\mathcal{X}\%\mathcal{T}$ 的值等于 0 时，M 将 NQ 和 OQ 均置为空，表明 M 将在新的一个周期内重新收集相应的元组。因为在上一个更新周期中，所有计算节点上的副本已经更新完毕。因此，直接从副本中恢复丢失的数据，而无需根据 $\langle NQ, OQ\rangle$ 信息来恢复丢失的计算结果。

3.4　容错并行查询过程

为了达到容错和高效查询处理的目的，需要将上述提出的容错策略有效地集成至并行查询过程中。REPS 方法容错查询的整个处理过程可归纳为算法 2。

算法 2　REPS 并行查询过程

1. M 采用分布式轮询映射策略将 e_{new} 映射至计算节点 P_i；
2. M 发送 $Q=\langle e_{new}, e_{old}, P_{new}, bRep\rangle$ 至所有计算节点；
3. if M 发送至某个计算节点 P_f 失效 then
4.　　根据算法 1 在新节点 P_s 中快速恢复丢失的数据；
5.　　M 发送 $\langle e_{new}, e_{old}, P_{new}, bRep\rangle$ 至 P_s；
6. foreach 任意的计算节点 P_i do
7.　　利用式(2)根据 e_{new} 和 e_{old} 更新集合 C_q, w_i；
8.　　P_i 计算支配概率 $P_i^d(e_{new})$ 值并将其发送至 P_{new}；
9.　　if bRep 为真 then
10.　　P_i 更新其位于其它计算节点上的所有副本；
11.　　P_i 通知 M 其所有的操作均已完成；
12. P_{new} 计算 $P_{sky}(e_{new})$ 值并将其返回至 M；
13. if M 未收到所有计算节点的操作完成消息 then
14.　　执行算法 3 的容错处理操作；

在 REPS 查询过程中，首先采用文献[18]中的分布式轮询映射策略，将新到达的元组 e_{new} 映射至某个计算节点(第 1 行)，然后发送四元组 $Q=\langle e_{new}, e_{old}, P_{new}, bRep\rangle$ 至所有的计算节点(第 2 行)。当接收到信息后，所有的计算节点即可并行计算对于 e_{new} 的支配概率，同时并行更新各主窗口中所维护的候选元组的 Skyline 概率等。Q 中的 P_{new} 主要用于告诉其它计算节点 e_{new} 属于哪个计算节点，从而能够将计算的对于 e_{new} 的支配概率值发送至 P_{new}；$bRep$ 用于告诉计算节点是否需要更新其副本信息。若 $bRep$ 为真，则当所有计算节点完成其计算任务后，均需要更新其在其它计算节点上维护的副本信息。此外，当 P_{new} 接收到 $P_{sky}(e_{new})$ 值后，需要将其发送至 M，以更新 NQ 中 e_{new} 的 Skyline 概率(第 12 行)。

在上述操作完成后，由于计算节点均需要通知 M 其操作已全部完成，因此可设定一个时间阈值 θ 来监控计算节点的状态。若在一定的时间间隔 θ 内，M 未收到来自其它计算节点的回复信息，则 M 发送探测消息至未返回信息的计算节点，并进一步采取相应的容错策略。特别地，θ 值的设置极为重要，若该值过大，则查询处理过程中的响应时间和处理时间

将会显著增加；若该值过小，将会导致很多失效情形的错误判定。在 REPS 方法中，主要根据在历史查询过程中每次查询更新的平均处理时间来对 θ 加以设定。在查询过程中，M 会记录未发生故障时每执行一次更新需要的平均时间；假定该平均时间为 t_{avg}，则将 θ 值设为 $\lambda \cdot t_{avg}$，其中 λ 是一个常整数值，在 REPS 方法中将 λ 的缺省值设为 3。

由于在查询处理的任何一个步骤中均可能发生故障，因此为了快速检测出故障，可直接将失效检测过程与具体的查询处理过程相结合。假定 *bSent* 表示计算节点是否发送支配概率值至 P_{new}，只有当 e_{new} 的支配概率成功发送至 P_{new} 时，*bSent* 的值才为真；*bSky* 表示 e_{new} 的 Skyline 概率是否已经计算完成，该值可从计算节点 M 处直接获得；*bRep* 表示是否需要进行副本更新，该值为真表明需要更新。特别地，只有当 e_{new} 的支配概率成功发送至 P_{new} 时，*bSent* 的值才为真。当 M 未收到来自全部计算节点的操作完成信息时，将启动新的容错处理机制，其具体的执行过程如算法 3 所示。

算法 3 细化容错处理算法

1. M 执行初始化工作；
2. foreach 任意的计算节点 $P_i(1 \leqslant i \leqslant n)$ do
3. if P_i 不为失效节点 then
4. 若 P_i 为 P_{new}，则计算 $P_{sky}(e_{new})$ 并将其返回 M；
5. 若 P_i 不为 P_{new} 且 bSent 为假，则发送支配概率至 P_{new}；
6. else
7. 根据算法 1 在替换节点 $P_{i'}$ 处快速恢复丢失的数据；
8. $P_{i'}$ 执行 Skyline 概率更新并计算 e_{new} 的支配概率；
9. 若 P_i 为 P_{new} 且 bSky 为假，计算 $P_{sky}(e_{new})$ 并将其返回 M；
10. 若 P_i 不为 P_{new}，则发送 e_{new} 的支配概率至 P_{new}；
11. if bRep 为真 then
12. 更新 P_i 在其它计算节点上的所有数据副本信息。

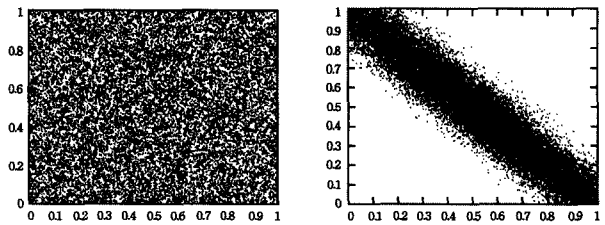
在算法 3 中首先执行初始化工作：首先，M 发送探测信息至未返回操作完成信息的所有计算节点，以确定这些计算节点的真实状态；其次，当发现失效节点后，M 选择系统中的空闲节点来替代失效节点，并通知系统中所有活跃的计算节点关于新的替代节点的信息。因此，所有由于发生故障而无法完成计算任务的计算节点均可继续执行查询处理操作。当以上操作完成后，M 发送〈NQ，OQ〉和〈e_{new}，e_{old}，P_{new}，*bRep*〉信息至新的替代节点，则该替代节点便能够根据算法 1 恢复全部丢失的数据信息。

对于所有失效计算节点的替代节点，首先需要恢复所有丢失的数据，然后根据 e_{new} 和 e_{old} 信息更新其维护的主窗口中候选元组的 Skyline 概率信息，同时计算 e_{new} 的支配概率值，而 P_{new} 节点则需要进一步计算 e_{new} 的 Skyline 概率值。其次，若失效节点即为 P_{new} 且 *bSky* 为假，则应该从其它计算节点处收集 e_{new} 的支配概率，同时计算 $P_{sky}(e_{new})$ 值并将其发送至 M；否则，P_i 应该发送 e_{new} 的支配概率至计算节点 P_{new}。当所有操作完成后，若 *bRep* 为真，则需要继续更新所有失效节点所对应的相关数据副本信息。

4 实验测试与分析

本文中的所有实验均部署于一个实际的数据中心环境中，该环境中包含了 3 个集群，每个集群包含 16 个同构的物理节点，且每个物理节点上部署两个虚拟节点，各虚拟节点配置为双核 2.6GHz Xeon CPU、4GB 内存、1TB 硬盘和千兆网卡。所有算法均由 Java 实现，并运行于 CentOS 操作系统。

实验中采用了多个合成数据集来产生数据流，且这些合成数据集主要包括两种最为常用的数据分布类型，即独立型数据集（简称 Indep）和反相关数据集（简称 Anti）[28]，如图 3 所示。此外，采用正态分布来随机产生并赋予每个合成数据对象一个存在概率值，以生成实验中的完整的流数据元组。其中，正态分布的均值和方差分别设为 0.65 和 0.35。实验中假定用户设定的缺省概率阈值和流元组维度分别为 0.3 和 4，同时全局滑动窗口的长度 $|W|$、计算节点数 n 和副本更新周期长度 $\mathcal{T}$ 的缺省值分别为 2M（即 2×10^6 个）、4 和 5。此外，由于实验中的各计算节点基本同构，因此假定所有计算节点维护的局部窗口长度相同，即 $|W_i|=|W|/n(1 \leqslant i \leqslant n)$。

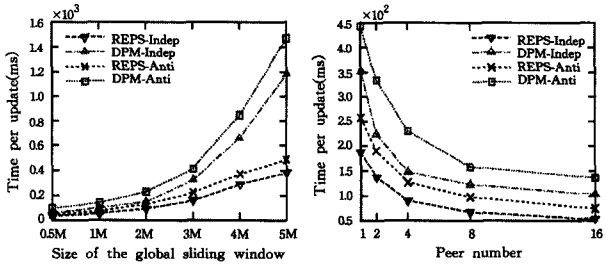


(a)独立型数据集(Indep)　(b)反相关数据集(Anti)

图 3　合成数据集类型示例

4.1 无节点失效时的性能

实验首先测试了在并行查询过程中无节点故障发生时 REPS 方法的查询处理性能，并将其与当前已有的高效并行查询方法 DPM(Distributed Parallel Model)[18] 进行对比。在 DPM 方法中，监控节点逐个交替地将数据按序映射至各计算节点，这能够有效地提高并行查询处理的效率。对于不同的全局滑动窗口规模和计算节点数目，测试的实验结果分别如图 4(a)和图 4(b)所示。



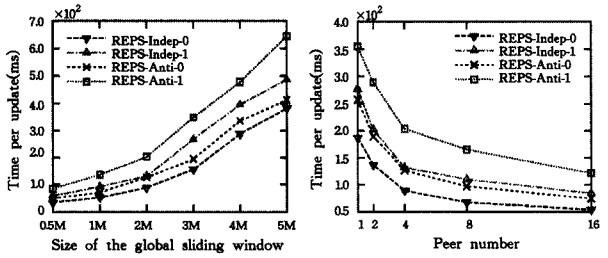
(a)全局滑动窗口规模对性能的影响　(b)计算节点数对性能的影响

图 4　无节点失效时并行查询处理的性能测试

由测试结果可知，当无计算节点故障发生时，REPS 方法的性能结果均显著优于 DPM 方法。尽管由于引入了多种容错机制和优化查询机制（例如仅考虑候选元组的更新等），导致 REPS 方法每次的更新时间有所增长，但是其并行处理的效率仍然很高，主要原因在于其并未引入过多的额外处理开销，在一定的程度上仍然能够满足快速查询的需求。

4.2 单节点失效时的性能

为了分析 REPS 方法的容错处理性能，测试了当单个计算节点失效时 REPS 方法每次查询更新的时间，并将其与无故障发生时的处理性能进行了对比，其在不同的滑动窗口规模和计算节点数时的测试结果分别如图 5(a)和图 5(b)所示。



(a)全局滑动窗口规模对性能的影响　(b)计算节点数对性能的影响

图 5　单节点失效时并行查询处理的性能测试

由测试结果可知，REPS 方法每次执行更新的时间随全局滑动窗口规模的增加而不断增加，其主要原因在于整体任务量的增加。然而，与无计算节点故障的情形相比，单个节点故障发生时其处理的性能有所下降。该结果是显然的，因为计算节点的失效将导致额外的处理开销，如恢复丢失的数据和恢复相应查询所需要的处理开销等。尽管如此，由图中结果可知，其引入的处理开销并非很大，可见即使在单个节点发生故障时，REPS 方法仍然能够快速有效地执行并行 Skyline 查询处理。

4.3　多节点失效时的性能

实验中专门测试了多个节点发生故障时 REPS 方法的处理性能，其实验结果如图 6 所示。实验中将参与并行处理的计算节点数设为 8，且图中给出的结果为更新 10000 次的平均结果。由图 6 可知，与单个计算节点发生故障时的情形相比，当两个或者多个计算节点发生故障时，其并行查询处理的性能有所下降。主要原因在于，相比于单个节点发生故障的情形，多个节点发生失效时的容错查询需要引入更多的数据恢复和查询恢复的时间。尽管在数据恢复和查询修复的过程中，处理多个节点故障时能够并行执行修复过程，然而其总体的修复时间显然比单个节点故障恢复的时间更长。然而，由图 6 可知，即使当全局滑动窗口的规模较大时，在每次更新处理的过程中其所增加的处理时间相对较小。

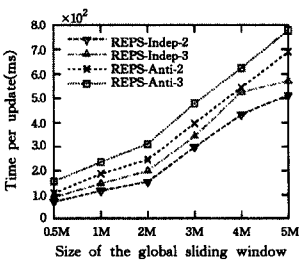


图 6　多节点失效时的查询性能

4.4　复制周期对性能的影响

为了测试不同副本更新周期对并行处理性能的影响，专门设计实验测试了当无节点故障发生时，不同副本更新周期长度对并行查询性能的影响，实验测试结果如图 7 所示。需要特别指出的是，实验中设定的副本更新周期为连续两次副本更新之间的时间间隔，而该时间间隔通过流元组的更新周期次数来设定，图 7 中的横坐标描述的副本更新周期长度值即为流元组查询更新的次数。

由图 7 可知，REPS 方法每次执行流元组更新的时间随着副本更新周期由 5 增加至 25 而不断减少。其主要原因在于，随着 $\mathcal{T}$ 值的不断增加，单次查询更新的平均处理时间相对减少。然而，随着 $\mathcal{T}$ 值的不断增加，恢复数据的处理时间也会不断增加。因此，在并行处理的过程中，应该选择一个适当的 $\mathcal{T}$ 值，使 REPS 方法既能满足用户对容错并行查询的需求，又能有较高的并行处理的效率。通常 $\mathcal{T}$ 的取值与节点故障发生频率的关系较大，当故障频繁发生时 $\mathcal{T}$ 可取较小的值，否则选择较大的值。

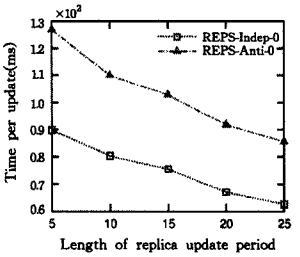


图 7　更新周期长度对性能的影响

结束语　针对并行概率流 Skyline 查询过程中由于发生故障而导致查询结果不准确和查询中断的问题，本文提出了一种基于复制的容错并行 Skyline 查询方法 REPS。该方法利用基于复制策略的容错机制保证了查询数据的可用性，同时对复制的内容、时机、副本的数目和副本放置的位置等方面进行了优化。在 REPS 中选择参与并行处理的计算节点作为副本节点，并对各计算节点上的多个副本进行层次化管理，通过选择优先级高的副本恢复数据，保证了数据恢复的高效性；同时将故障检测、丢失数据恢复和查询过程恢复贯穿于整个查询过程，减少了容错处理的额外通信开销和计算开销，并实现了快速的查询恢复过程。实验结果表明，REPS 方法能够快速检测故障并恢复并行查询处理过程，不仅在无故障发生时具有高效的并行查询处理能力，而且在单个节点甚至多个节点失效时，仍然能够保持较高的容错并行查询处理能力，能够满足用户的查询处理需求。

参 考 文 献

[1] Li X, Wang Y, Li X, et al. Parallelizing Skyline Queries over Uncertain Data Streams with Sliding Window Partitioning and Grid Index [J]. Knowledge and Information Systems(KAIS), 2014, 41(2): 277-309

[2] 王意洁，李小勇，杨永滔，等. 不确定 Skyline 查询技术研究 [J]. 计算机研究与发展，2012，49(10)：2045-2053
Wang Yi-jie, Li Xiao-yong, Yang Yong-tao, et al. Research on uncertain skyline query processing techniques [J]. Computer Research and Development, 2012, 49(10): 2045-2053

[3] Chomicki J, Ciaccia P, Meneghetti N. Skyline queries, front and back [J]. ACM SIGMOD Record, 2013, 42(3): 6-18

[4] Lu H, Zhou Y, Haustad J. Efficient and scalable continuous skyline monitoring in two-tier streaming settings [J]. Information Systems, 2013, 38(1): 68-81

[5] Huang Z, Sun S, Wang W. Efficient mining of skyline objects in subspaces over data streams [J]. Knowledge and information systems(KAIS), 2010, 22(2): 159-183

[6] Sun S, Huang Z, Zhong H, et al. Efficient monitoring of skyline queries over distributed data streams [J]. Knowledge and Information Systems(KAIS), 2010, 25(3): 575-606

[7] Tao Y, Papadias D. Maintaining sliding window skylines on data streams [J]. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(3): 377-391

[8] Lin X, Yuan Y, Wang W, et al. Stabbing the sky: Efficient skyline computation over sliding windows [C]// Proceedings of the IEEE International Conference on Data Engineering (ICDE). 2005: 502-513

(下转第 264 页)

rithm for attribute reduction[J]. Control and Decision, 2011, 26(4): 495-500

[13] 杨传健,葛浩,李龙澍. 垂直划分二进制可分辨矩阵的属性约简[J]. 控制与决策,2013,28(4):563-568
Yang Chuan-jian, Ge Hao, Li Long-shu. Attribute reduction of vertically partitioned binary discernibility matrix[J]. Control and Decision, 2013, 28(4): 563-568

[14] 杨明. 一种基于改进差别矩阵的属性约简增量式更新算法[J]. 计算机学报,2007,30(5):815-822
Yang Ming. An incremental updating algorithm for attribute reduction based on improved discernibility matrix[J]. Chinese Journal of Computers, 2007, 30(5): 815-822

[15] 刘少辉,盛秋戬,吴斌,等. Rough 集高效算法的研究[J]. 计算机学报,2003,26(5):524-529
Liu Shao-hui, Sheng Qiu-jian, Wu Bin, et al. Research on efficient algorithms for rough set methods[J]. Chinese Journal of Computers, 2003, 26(5): 524-529

[16] 刘少辉,盛秋戬,史忠植. 一种新的快速计算正区域的方法[J]. 计算机研究与发展,2003,40(5):637-642
Liu Shao-hui, Sheng Qiu-jian, Shi Zhong-zhi. A new method for fast computing positive region[J]. Journal of Computer Research and Development, 2003, 40(5): 637-642

[17] 徐章艳,刘作鹏,杨炳儒,等. 一个复杂度为 max(O(|C||U|), O(|C^2|U/C|)) 的快速属性约简算法[J]. 计算机学报,2006,29(3):391-399
Xu Zhang-yan, Liu Zuo-ping, Yang Bing-ru, et al. Quick attribute reduction algorithm with complexity of max(O(|C||U|), O(|C|^2)|U/C|))[J]. Chinese Journal of Computers, 2006, 29(3): 391-399

[18] 刘勇,熊蓉,褚健. Hash 快速属性约简算法[J]. 计算机学报,2009(8):1493-1499
Liu Yong, Xiong Rong, Chu Jian. Quick attribute reduction algorithm with hash[J]. Chinese Journal of Computers, 2009, 32(8): 1493-1499

[19] 葛浩,李龙澍,杨传健. 基于冲突域的高效属性约简算法[J]. 计算机学报,2012,35(2):342-350
Ge Hao, Li Long-shu, Yang Chuan-jian. An efficient attribute reduction algorithm based on conflict region[J]. Chinese Journal of Computers, 2012, 35(2): 342-350

[20] 葛浩,李龙澍,杨传健. 一种核属性快速求解算法[J]. 控制与决策,2009,24(5):738-742
Ge Hao, Li Long-shu, Yang Chuan-jian. Quick algorithm for computing core attribute[J]. Control and Decision, 2009, 24(5): 738-742

[21] 葛浩,杨传健,李龙澍. 一种高效的核属性求解算法[J]. 计算机工程与应用,2010,46(26):138-141
Ge Hao, Li Long-shu, Yang Chuan-jian. Efficient algorithm for computing core attributes[J]. Computer Engineering and Applications, 2010, 46(26): 138-141

[22] Hu F, Wang G, Xia Y. Attribute core computation based on divide and conquer method[M]// Rough Sets and Intelligent Systems Paradigms. Springer, 2007: 310-319

(上接第 230 页)

[9] Morse M, Patel J, Grosky W. Efficient continuous skyline computation [J]. Information Sciences, 2007, 177(17): 3411-3437

[10] Zhang Z, Cheng R, Papadias D, et al. Minimizing the communication cost for continuous skyline maintenance [C]// Proceedings of the ACM International Conference on Management of Data (SIGMOD). 2009: 495-508

[11] Xin J, Wang G, Chen L, et al. Continuously maintaining sliding window skylines in a sensor network [C]// Proceedings of the International Conference on Database Systems for Advanced Applications(DASFAA). 2007: 509-521

[12] Kontaki M, Papadopoulos A N, Manolopoulos Y. Continuous Top-k Dominating Queries [J]. IEEE Transactions on Knowledge and Data Engineering(TKDE), 2012, 24(5): 840-853

[13] 孙圣力,戴东波,黄震华,等. 概率数据流上 Skyline 查询处理算法 [J]. 电子学报,2009,37(2):285-293
Sun Sheng-li, Dai Dong-bo, Huang Zhen-hua, et al. Algorithm on computing Skyline over probabilistic data stream [J]. Acta Electronica Sinica, 2009, 37(2): 285-293

[14] Zhang W, Lin X, Zhang Y, et al. Probabilistic skyline operator over sliding windows [C]// Proceedings of the IEEE International Conference on Data Engineering (ICDE). 2009: 1060-1071

[15] Lian X, Chen L. Monochromatic and bichromatic reverse skyline search over uncertain data [C]// Proceedings of the ACM International Conference on Management of Data(SIGMOD). 2008: 213-226

[16] Ding X, Lian X, Chen L, et al. Continuous monitoring of skylines over uncertain data streams [J]. Information Sciences, 2012, 184(1): 196-214

[17] Wang Y, Li X, Li X, et al. A survey of queries over uncertain data [J]. Knowledge and Information Systems(KAIS), 2013, 37(3): 485-530

[18] Li X, Wang Y, Li X, et al. Parallel skyline queries over uncertain data streams in cloud computing environments [J]. International Journal of Web and Grid Services(IJWGS), 2014, 10(1): 24-53

[19] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters [C]// Proceedings of the USENIX Symposium on Operating System Design and Implementation (OSDI). 2004: 137-150

[20] MacCormick J, Murphy N, Najork M, et al. Boxwood: Abstractions as the Foundation for Storage Infrastrucure [C]// Proceedings of the USENIX Symposium on Operating System Design and Implementation(OSDI). 2004: 105-120

[21] Wu P, Zhang C, Feng Y, et al. Parallelizing skyline queries for scalable distribution [C]// Advances in Database Technology (EDBT): Proceedings of the International Conference on Extending Database Technology. Springer, 2006: 112-130

[22] Wang S, Ooi B, Tung A, et al. Efficient skyline query processing on peer-to-peer networks [C]// Proceedings of the IEEE International Conference on Data Engineering (ICDE). 2007: 1126-1135

[23] 王媛,王意洁,邓瑞鹏,等. 云计算环境下的容错并行 Skyline 查询算法研究 [J]. 计算机科学与探索,2011,5(9):804-814
Wang Yuan, Wang Yi-jie, Deng Rui-peng, et al. Fault-tolerant parallel skyline computation in cloud computing environment [J]. Journal of Frontiers of Computer Science and Technology, 2011, 5(9): 804-814