Journal of Southeast University (English Edition) Vol. 25, No. 4, pp. 456 – 459 Dec. 2009 ISSN 1003—7985

# Min-wise hash function-based sampling over distributed data streams

Chong Zhihong Ni Weiwei Xu Lizhen Lü Jianhua Xie Yinghao

(School of Computer Science and Engineering, Southeast University, Nanjing 210096, China)

**Abstract:** In order to avoid the redundant and inconsistent information in distributed data streams, a sampling method based on min-wise hash functions is designed and the practical semantics of the union of distributed data streams is defined. First, for each family of min-wise hash functions, the data with the minimum hash value are selected as local samples and the biased effect caused by frequent updates in a single data stream is filtered out. Secondly, for the same hash function, the sample with the minimum hash value is selected as the global sample and the local samples are combined at the center node to filter out the biased effect of duplicated updates. Finally, based on the obtained uniform samples, several aggregations on the defined semantics of the union of data streams are precisely estimated. The results of comparison tests on synthetic and real-life data streams demonstrate the effectiveness of this method.

**Key words:** data streams; aggregation; min-wise hashing

With the advances in information technology, data streams are generated in huge amounts of volume. Examples include IP address pairs in network transmission, detected information in sensor networks, etc. As a variant of a data stream model, distributed data streams (DDSs) emerge as a promising research direction in data streams[1]. Massive data streams generated by physically distributed sources need to be processed in a distributed fashion. The prominent feature of this model is that multiple update streams are fed into multiple runs of an algorithm distributed on different nodes. This setup is used in Lucent's Interpret Net and Cisco's Net flow Network to monitor products[2].

In the DDS model, the aggregate statistics, such as min(), max(), average(), and sum(), are of great interest. The nature of the distributed data streams is that a large quantity of information is flooded at a very high rate, while only limited processing resources such as memory are available. Therefore, the computation of aggregate functions on the union of the data streams poses a great challenge. Intuitively, we can maintain uniform samples of each update stream and use the samples to estimate the aggregate function. A problem with this simple solution, however, arises. For example, there are two distributed update streams with some duplicate entities crossing. If each node uniformly samples an entity with the same probability, then the probability of sampling the duplicates is doubled, which biases the samples for the union of two update streams.

In this paper, a novel sampling technique is introduced to obtain uniform samples with a fixed number on the union of the update streams and to establish the coordination between distributed samples. The sampling technique differs from previous works[1] in that this approach aims at a turnstile model, the most general model in practice, rather than binary series. We also define a special semantics for the union of update streams and apply it to distributed systems. Combined with the min-wise hash sampling technique, it can be applied to filter out duplicates which are ubiquitous in distributed systems. Furthermore, we provide solutions to several important aggregate functions in distributed systems rather than only estimating cardinality.

## 1 Aggregation of Union of Update Streams

The union of the update streams on two nodes $N_i$ and $N_j$ is the union of their corresponding update vectors $\boldsymbol{V}_i$ and $\boldsymbol{V}_j$. Without loss of generality, these values of the same entity are assumed to be recorded in the same slots on different vectors. We define the binary operations on two update vectors as follows:

1) $\boldsymbol{V}_i \cup \boldsymbol{V}_j$ represents the union addition of $\boldsymbol{V}_i$ and $\boldsymbol{V}_j$ and can be computed by $\boldsymbol{V}_i[x] + \boldsymbol{V}_j[x]$ for all $x$.

2) $\boldsymbol{V}_i \vee \boldsymbol{V}_j$ represents the union maximum of $\boldsymbol{V}_i$ and $\boldsymbol{V}_j$ and can be computed by $\max\{\boldsymbol{V}_i[x], \boldsymbol{V}_j[x]\}$ for all $x$.

3) $\boldsymbol{V}_i \wedge \boldsymbol{V}_j$ represents the union minimum of $\boldsymbol{V}_i$ and $\boldsymbol{V}_j$ and can be computed by $\min\{\boldsymbol{V}_i[x], \boldsymbol{V}_j[x]\}$ for all $x$.

For example, nodes $N_i$ and $N_j$ record the update states of two update streams in two vectors $\boldsymbol{V}_i = \langle$ null, 188, 288, null$\rangle$, $\boldsymbol{V}_j = \langle 18, 199,$ null, $28\rangle$. Then, $\boldsymbol{V}_i \cup \boldsymbol{V}_j = \langle 18, 199 + 188, 288, 28\rangle$, $\boldsymbol{V}_i \vee \boldsymbol{V}_j = \langle 18, 199, 288, 28\rangle$, $\boldsymbol{V}_i \wedge \boldsymbol{V}_j = \langle 0, 188, 0, 0\rangle$.

Note that the union of two vectors is still a vector. An aggregate function applied to a vector can perform the operation on the values which are not null.

There are several types of queries that users or applications may pose on distributed update streams. Examples include joins[3], norm computations[4], and quantile estimation[5]. One of the most fundamental queries on data streams is the aggregate query. In this paper, we focus on the estimation of the aggregate function on the union of the update streams. For example, for the aggregate function sum() on the union of the two vectors $\boldsymbol{V}_i$ and $\boldsymbol{V}_j$, we obtain $\text{sum}(\boldsymbol{V}_i) = 188 + 288$, $\text{sum}(\boldsymbol{V}_j) = 18 + 199 + 28$, $\text{sum}(\boldsymbol{V}_i \cup \boldsymbol{V}_j) = 18 + (199 + 188) + 288 + 28 = \text{sum}(\boldsymbol{V}_i) + \text{sum}(\boldsymbol{V}_j)$, $\text{sum}(\boldsymbol{V}_i \vee \boldsymbol{V}_j) = 18 + 199 + 288 + 28 \neq \text{sum}(\boldsymbol{V}_i) + \text{sum}(\boldsymbol{V}_j)$, $\text{sum}(\boldsymbol{V}_i \wedge \boldsymbol{V}_j) = 0 + 188 + 0 + 0 \neq \text{sum}(\boldsymbol{V}_i) + \text{sum}(\boldsymbol{V}_j)$.

Subsequently, we can obtain $\text{sum}(\boldsymbol{V}_i \cup \boldsymbol{V}_j)$ by $\text{sum}(\boldsymbol{V}_i)$

**Received** 2009-04-14.

**Biography:** Chong Zhihong (1969—), male, doctor, lecturer, chongzhihong@seu.edu.cn.

**Foundation items:** The National Natural Science Foundation of China (No. 60973023, 60603040), the Natural Science Foundation of Southeast University (No. KJ2009362).

**Citation:** Chong Zhihong, Ni Weiwei, Xu Lizhen, et al. Min-wise hash function-based sampling over distributed data streams[J]. Journal of Southeast University (English Edition), 2009, 25(4): 456 – 459.

$+\text{sum}(\boldsymbol{V}_j)$. However, it is not easy to compute $\text{sum}(\boldsymbol{V}_i \vee \boldsymbol{V}_j)$ or $\text{sum}(\boldsymbol{V}_i \wedge \boldsymbol{V}_j)$ when the vector is too great to be maintained entirely on a node, let alone transmit it from one node to another. Therefore, estimation or sampling techniques are used to solve this problem. Because $\boldsymbol{V}_i \vee \boldsymbol{V}_j$ and $\boldsymbol{V}_i \wedge \boldsymbol{V}_j$ have the similar properties, we focus on estimating the aggregate functions on $\boldsymbol{V}_i \vee \boldsymbol{V}_j$.

Suppose that the domain set of all entities is $X$ whose value range is $[-M/2, M/2]$ and reasonably assumed to be not too great. As for the turnstile model, an update $u \in X \times [-M/2, M/2]$ in the form of $(x, \Delta)$ is interpreted as updating the value of the entity $x$ by $\Delta$, i.e., $\boldsymbol{V}_i[x] = \boldsymbol{V}_i[x] + \Delta$. Note that the initial value $\boldsymbol{V}_i[x]$ for the entity $x$, which is not yet observed, is null.

Let $|\boldsymbol{V}|$ be the number of slots which are not null. An aggregate function g() applied to a vector $\boldsymbol{V}$ is considered as a map $g$: $\boldsymbol{V} \rightarrow \mathbf{R}$. For example, if $g()$ is an average aggregate function, then $g(\boldsymbol{V}) = \sum_{\boldsymbol{V}[x] \neq \text{null}} \boldsymbol{V}[x] / |\boldsymbol{V}|$, which gives the average of a vector or an update stream.

Instead of maintaining an exact vector of $O(|X|)$ slots, it is highly desirable to obtain a small set of samples $\boldsymbol{V}'_i$ in place of $\boldsymbol{V}_i$. When these samples are combined(denoted as $\boldsymbol{V}'_i + \boldsymbol{V}'_j$), three problems exist as follows: 1) How to generate a small set of samples $\boldsymbol{V}'_i$ of $\boldsymbol{V}_i$ on each node $N_i$ fast and directly from the raw update stream $U_i$; 2) How to compute $\sum \boldsymbol{V}'_i$ to approximate $\sum \boldsymbol{V}_i$ when a receiver node obtains multiple sets of samples $\boldsymbol{V}'_i$; 3) How to make an accurate estimation $g(\sum \boldsymbol{V}'_i)$ for the true aggregate function $g(\sum \boldsymbol{V}_i)$.

## 2 Preliminaries

A central tool in our algorithms is the min-wise hash[6]. Let $\pi$ be a randomly chosen permutation over the set $X$. Then $\pi$ is a mapping $\pi$: $X \rightarrow 1, 2, \ldots, |X|$. Its inverse mapping $\pi^{-1}(i)$ is an entity $x$ in $X$ which is permuted in the position $i$ in terms of $\pi$.

Assuming that $X_i$ is a subset of $X$, the min-wise hash $h_\pi(X_i)$ for a given permutation $\pi$ can be defined as $h_\pi(X_i) = \min\{\pi(x) \mid x \in X_i\}$. Because the ideal family of min-wise hash functions are obtained on all permutations over $X$, it requires $O(|X|\log|X|)$ memory, which is normally beyond the processing ability of most systems. Alternatively, we can use the $\varepsilon'$-min-wise independent hash in a family of hash functions $H$. A family of hash functions $H \subset X \rightarrow X$ is called $\varepsilon'$-min-wise independent if $\Pr_{h \in H}\{h(x) < h(X)\} = 1/(|Y|+1)(1 \pm \varepsilon')$ holds for any $Y \subset X$ and $x \in X - Y$. Then the property $\Pr_{h \in H}\{h(x) = h(X_i)\} = 1/|X_i|(1 \pm \varepsilon')$ can be obtained for any $x \in X_i$.

In this algorithm, we need to estimate the number of distinct entities and directly apply several existing algorithms[7].

## 3 Estimation on Average of Union of Update Streams

The average of the union of update streams is estimated; i.e., $g(\vee \boldsymbol{V}_i)$ is estimated by $g'(\sum \boldsymbol{V}_i)$, where $g()$ is the average aggregate function, and $g'()$ is the approximation of $g()$. The algorithm processes three events as shown in Tab. 1.

**Tab. 1** Algorithms for each event

| Event | Update event | Combination event | Estimation event |
|---|---|---|---|
| Pre-condition | {input $u_n$} | {collect local samples} | {update global samples} |
| Process | sum up $\Delta$ for the minimum hash value for each hash function | Sum up $\Delta$ for each hash function with the same minimum hash value | Average all samples |
| Post-condition | {updated local samples} | {updated global samples} | {estimated aggregation} |

1) The update event on each node $N_i$

On each node $N_i$, we choose $\xi$ min-wise hash functions $h_1(), \ldots, h_\xi()$. For each hash function $h_j$, a tuple $\boldsymbol{V}'_i[j] = \langle m, v \rangle$ is maintained, where $\boldsymbol{V}'_i[j].m = \min\{h_j(u_n.x)\}$, and $\boldsymbol{V}'_i[j].v$ is the total update value for the entity $x$ whose $j$-th min-hash value is $\boldsymbol{V}'_i[j].m$. The samples obtained at local nodes are stored in the array $\boldsymbol{V}'_i$ of $\xi$ tuples of the form $\langle m, v \rangle$.

2) The combination event on the central node

When the central node obtains two sets of local samples $\boldsymbol{V}'_i$ and $\boldsymbol{V}'_j$, the node merges them into $\boldsymbol{V}'_i + \boldsymbol{V}'_j$ by selecting the sample corresponding to the smaller min-wise hash value for the same hash function from each pair of $\boldsymbol{V}'_i[l]$ and $\boldsymbol{V}'_j[l]$. When the two min-wise hash values are equal, the two samples are added because they are for the same entity. For more than two local samples, the above process is recursively applied.

3) The estimation event on the central node

The average function on the global samples is applied and the estimated aggregation value is outputted.

The size of $\sum \boldsymbol{V}'_i$ is $\xi$, where $\xi$ is determined by Hoeffding's inequality[8].

**Lemma 1** $\sum \boldsymbol{V}'_i$ is a uniform sampling of $\vee \boldsymbol{V}_i$.

The error of this approach can be statistically estimated by the theorem below.

**Theorem 1** Given $\xi \geqslant (M^2 \ln 1/\delta)/(2\varepsilon^2)$ with any error parameter $\varepsilon$ and confidence parameter $\delta$, we can obtain $\Pr\{\text{Avg}(\sum \boldsymbol{V}'_i) - g(\vee \boldsymbol{V}_i) \mid < \varepsilon\} \geqslant 1 - \delta$ if $g()$ is an average aggregate function.

Heoffding's inequality and lemma 1 can deduce this theorem. It is worth noting that some duplicates can be generated by a loop in the topology and this method can filter out them easily.

## 4 Performance Study

Extensive experiments are conducted on a PC with 2.4 GHz CPU, 1 GB memory, running Microsoft Windows XP Professional. The development environment is Visual C++ 6.0. The PC is used to simulate the environment of up to 1 000 nodes. Each node has a large volume of data streams. Therefore, only a small set of samples can be generated for each node.

Update streams and online news are generated from synthetic data streams of the Zipf distribution and the Reuters real-time data feed, respectively. The Zipf distribution is ubiquitous in the real world and used in many experiments for the frequency estimation of a stream. In the Zipf distribution, the parameter $Z$ controls the distribution of the frequency; e. g., a high value of $Z$ means a relatively more skewed distribution. In this experiment, the largest number of distinct entities possibly appearing in a Zipf stream is fixed to be $2 \times 10^7$. Subsequently, a Zipf sequence is converted into a stream of updates $(x_1, \Delta_1)$, …, $(x_n, \Delta_n)$, …. This means that the entity $x_n$ is updated by $\Delta_n$ at the $n$-th update, where $\Delta_n$ is randomly chosen from the domain $\Delta$. The expected value $E(\Delta)$ of each update is controlled by skewing the value of $\Delta$. The influence of the length $L$ of an update stream is also considered. A data set can be characterized by these four factors. For example, $Z1.5\Delta100E(\Delta)10L10^6$ denotes a stream of $10^6$ updates with a Zipf of 1.5, an expected value of 10 and an update range of 100.

The online Reuters news data contains 365 288 news stories and 100 672 866 words with duplicates. The size of the data is 650 MB. The data set is processed by removing the common stop-words and stemming. The resulting data set is converted into an update stream in the same way as the construction of a synthetic update stream.

Logically, a distributed system can be stratified into rings based on the diffusion path of an aggregate demand. The center of the rings is the node which initially issues the aggregate demand. The first ring consists of the direct neighbors of the center node. The nodes on the first ring directly receive the aggregate demand from the center node. The neighbors of those nodes on the first ring form the second ring, and so on. The samples on each node are reversely transmitted along the demand diffusion path. Loops and duplicates inevitably occur in some regions of the rings. A loop can be logically identified by the appearance of the same node (inside the dash-line confined region) on different rings.

A small approximately min-wise independent family of hash functions is implemented to generate $\xi$ min-wise hash functions, where $\xi = (M^2 \ln(1/\delta))/(2\varepsilon^2)$ [6]. The estimation error rate $\alpha$ can be measured by $\alpha = |g() - g'()|/|g()|$.

The estimation precision is affected by four factors as follows: 1) The properties of the update streams characterized by data parameters; 2) The underlying topology of the network; 3) The configuration update on each node; 4) The parameter $\xi$ ($10^4$ as default) which determines the number of the samples.

Changing the number of nodes from 100 to 1 000, the error rate is fairly stable (see Fig. 1). That is, this algorithm has a very stable performance in cases with a varying number of nodes.

We compare our algorithm, denoted as H-sampling, with the coordinated sampling[1], denoted as C-sampling for the two-node case with $10^3$ and $10^4$ samples. To study the influence of the duplicates on the estimation in the distributed systems, some entities are allocated to more than one node. The percentage of duplicates varies from 0% to 100% with an interval of 20%. The testing results are shown in Fig. 2.

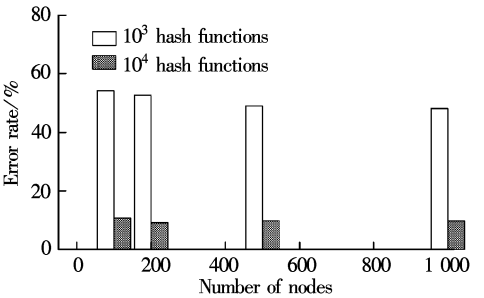


**Fig. 1** Number of nodes vs. error rate for estimating average functions

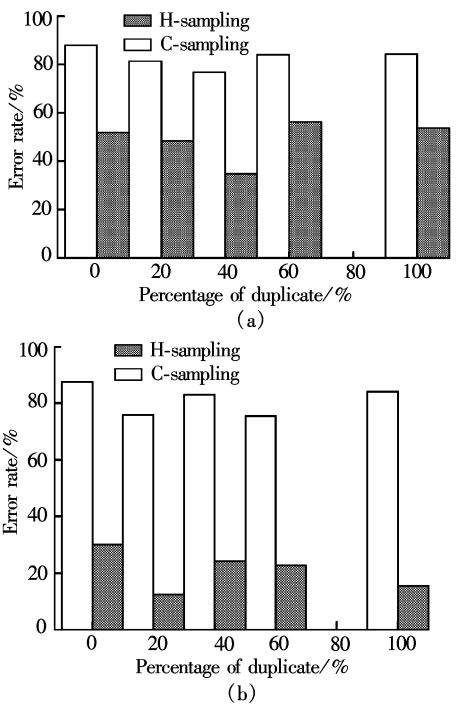


**Fig. 2** Comparison between H-sampling and C-sampling. (a) $10^3$ hash functions; (b) $10^4$ hash functions

It can be seen that H-sampling has a lower error rate than that of C-sampling. The estimation of C-sampling is always less than the exact average because some updates are skipped before they are sampled. However, the estimation of H-sampling fluctuates around the exact average. The error rate of C-sampling does not change much with the number of samples because no matter how many samples there may be, some updates are always missed before they are sampled. Nevertheless, H-sampling can greatly improve the estimation quality with more samples.

This algorithm is also applied to the Reuters online news messages. First, the online news is converted to form an update stream. The number of min-wise hash functions $\xi$ is set to be $10^3$, much less than the default value $10^4$. Then the same piece of news is randomly allocated to the configurations with 10 to 20 nodes, which can simulate the duplicate effect. The duplicates are ubiquitous in distributed systems such as P2P where multiple peers have the same piece of information. The testing results are shown in Tab. 2. It can be seen that this algorithm can achieve a low error rate even with a Zipf parameter of 2. Neither the number of du-

plicates nor the number of nodes can change the resulting estimation error.

**Tab. 2** Real streams with $10^3$ hash functions

| Node | Average | Estimation | Error/% | Time/s |
|---|---|---|---|---|
| 100 | 215.68 | 181.23 | 16.08 | 54 |
| 200 | 222.57 | 234.56 | 5.14 | 152 |
| 500 | 244.75 | 234.56 | 4.39 | 376 |
| 1 000 | 267.14 | 272.13 | 1.82 | 758 |

## 5 Conclusion

The fundamental problem of how to obtain uniform samples over distributed update streams is studied. Based on the min-wise hash functions, we maintain a fixed number of uniform samples on each node, and merge them to form uniform samples over the union of the update streams. This algorithm is robust in the presence of various physical and logical configurations in terms of the number of nodes, rings, loops and duplicates. Its error rate is small even if only a small set of samples are generated. The experiments on synthetic and real-life data sets show that this algorithm can always obtain uniform samples of the union of distributed update streams, which leads to more accurate estimations of aggregate functions than previous approaches.

## References

[1] Gibbons P B, Tirthapura S. Estimating simple functions on the union of data streams [C]//*Annual ACM Symposium on Parallel Algorithms and Architectures*. Crete Island, Greece, 2001: 281 - 290.

[2] Ganguly S, Garofalakis M, Rastogi R. Processing set expressions over continuous update streams [C]//*Proceedings of the ACM SIGMOD International Conference on Management of Data*. San Diego, CA, USA, 2003: 265 - 276.

[3] Ganguly S, Garofalakis M, Kumar A, et al. Join-distinct aggregate estimation over update streams [C]//*Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. Baltimore, Maryland, USA, 2005: 259 - 270.

[4] Indyk P. Stable distributions, pseudorandom generators, embeddings and data stream computation [C]//*Annual Symposium on Foundations of Computer Science Proceedings*. New York, NY, USA, 2000: 189 - 197.

[5] Gilbert A C, Kotidis Y, Muthuukrishnan S, et al. How to summarize the universe: dynamic maintenance of quantiles [C]//*Proceedings of the* 28*th Annual International Conference on Very Large Data Bases*. Hong Kong, China, 2002: 454 - 465.

[6] Broder A Z, Charikar M, Frieze A M, et al. Min-wise independent permutations [C]//*Conference Proceedings of the Annual ACM Symposium on Theory of Computing*. Dallas, Texas, USA, 1998: 327 - 336.

[7] Flajolet P, Martin G N. Probabilistic counting algorithms for data base applications [J]. *Journal of Computer and System Sciences*, 1985, **31**(2): 182 - 209.

[8] Hoeffding W. Probability inequalities for sums of bounded random variables [J]. *Journal of the American Statistical Association*, 1963, **58**(1): 13 - 30.

# 分布数据流上基于 Min-wise 散列函数的采样

崇志宏　倪巍伟　徐立臻　吕建华　谢英豪

（东南大学计算机科学与工程学院，南京 210096）

**摘要：**针对分布数据流中存在的冗余和不一致信息问题，提出了一种基于 Min-wise 散列的采样方法，并定义了反映应用需求的分布数据流并的语义. 首先，对于每一族 Min-wise 散列函数选取具有最小散列值的数据作为局部样本，滤除单个数据流中的频繁更新对采样偏斜的影响. 然后，对于相同散列函数产生的样本选取具有最小散列值的样本作为全局样本，完成局部样本集在中心节点的合并，滤除在分布节点上的重复更新对样本偏斜的影响. 最后，利用获得的均匀样本集，在多种数据流并的语义上精确估计聚集函数的值. 基于人造数据和真实数据的对比试验验证了该方法的有效性.

**关键词：**数据流；聚集；Min-wise 散列

**中图分类号：**TP392