Optimization of Resources Discovery in Grid Computing Using Bloom Filter

Raziye Tavuseh  
Computer Engineering Department,  
Azad Islamic University,  
Borujerd Unit, Borujerd, Iran  
raziehtawose@yahoo.com

Mahmood Ahmadi  
Computer Engineering Department,  
Razi University of Kermanshah,  
Kermanshah, Iran.

Abstract: In this paper, the counting Bloom filter is used to save resource specifications in grid information services. It is also used to inquire requested resources by users with a proper technique. The degree of effectiveness in term of mean time spent on grid information services for searching and discovering requested resources and number of bytes transferred over the network is evaluated. Results show that mean time spent on searching and discovering requested resource as well as the number of bytes transferred over the network for resources requested by users are about 80% and 78%, in proposed method, less than those in the case not using Bloom filter. Therefore, using the Bloom filter highly increases the speed of register, request, search, and discovery of resources. Furthermore, it can be very useful and efficient in their optimization when a lot resources are available, a lot information need to be saved, and the number of requests are increased.

Keywords: Grid computing, Grid information services, Discovery of resources in grid, Bloom filter, false positive, false Negative

I. INTRODUCTION

Internet world makes other computer information accessible. In fact, grid computing with help of Internet has provided a ground through which hardware resources of other systems can be used. Grids use resources of computers connected to network and are capable of doing highly complex computations easily by using these resources, force resultant. As computational resources and networks are developed in grids, optimal and efficient utilization of resources is outlined as one of highly important issues in grids. Grid resources include processors, data storages, and services of applicable programs. Any device connected to grid network is defined as one grid node. Most grid nodes are distributed geographically, heterogeneous in forms of structure and management, and joined to each other by means of links with differing efficiency and reliability. Discovering resources requested by users is a crucial requirement to support location independent computations. Since resources in grid systems are extremely heterogeneous and very dynamic in terms of stability, resource discovery becomes a time consuming process, which can reduce total system efficiency [6]. In this direction, a mechanism which can discover resources requested by users with the least traffic and appropriate speed is more effective. Quick discovery of accessible and retainable resources and dissemination of information on resource status effectively are among key requirements for grids, obtaining optimal productivity from systems and balancing loads among network computers. For mechanism of grid resource discovery, grid information service (GIS) is the most important part being used. As an underlying component, GIS is responsible for gathering and distributing information about grids. GIS provides tools in order to register resources, to inquire databases, and to delete lost nodes [7]. So information services are a vital part of each grid software foundation in order to create fundamental mechanisms for discovering and monitoring, planning and matching program behaviors [8]. In this paper, the data structure of Bloom filter is used to send information and to inquire resources using a suitable technique in grid; and its effects on inquiry speed, amount of bandwidth utilized, hence on optimization of resources discovery mechanism are investigated. In this method, a 2-level hierarchical structure, each level using counting Bloom filter to store resource information, is used in order to save and inquire resources. Hash functions to locate positions related to resource Information within Bloom filter, are used by resource nodes to register and by user nodes to inquire resources. But the data structure of counting Bloom filter is saved at 2 levels, in local grid information services at the 1st level and in GIS at the second one. In order to examine efficiency of proposed method compared to the case not using Bloom filter, time duration of resource discovery and amount of bandwidth used were measured in 2 modes. Results indicate that, in proposed method, mean time spent on searching and discovering requested resources is about 80% and the number of bytes transferred over the network for requesting resources needed by users and sending response to them is about 78% less than those in the case
not using Bloom filter, therefore making use of the data structure of Bloom filter along with applying an appropriate technique can be helpful to optimize resource register, request, search, and discovery processes.

In this direction, remaining sections of present paper are organized as follows: Section 2 presents Related works done in the field of discovering grid resource; Section 3 introduces the data structure of Bloom filter; Section 4 outlines the method proposed to architecture of storing resource information in grid information services and inquiry of grid resources using the data structure of Bloom filter; Section 5 evaluates proposed method; and section 6 provides conclusions.

II. REVIEW OF RELATED WORKS

[1] has used Resource Description Framework metadata (RDF metadata) in order to display and code both resources and requests. Nodes of local information are connected to each other, forming a P2P overlay. Their strategy is to extract structural blocks from RDF metadata, then, to summarize them into a compact structure called triple filter, which contains 3 Bloom filter for subject, predicate and object. A RDF triple can be hashed into these 3 filters. In addition to resource information, filters keep the distance to resource, which directs the progress of inquiry. If inquiry passes through a node's filters, then it makes progress toward that node. In any case, since matched elements may belong to different resources, this model cannot guarantee that inquiry can be responded while progressing on the path. In [2], a model has been outlined which mainly relies on combined GIS and Bloom filter on the basis of findings of grid P2P system based on distributed hash table. Strategy of matching special elements from Bloom filter was used to Reduce time complexity from $O(M^2)$ to $O(1)$, with M is the number of resources located on a node which the site of targeted resource has been determined. Nodes of information servers have been organized within a chord loop, which use Bloom filter in order to store and manage their registered resources and cache information recorded in their neighboring servers clockwise. They frequently update local Bloom filter as required and send to neighboring servers update messages in order to inform them, sending messages in the form of Compressed Bloom filter in this process in order to reduce network traffic. In [3], services were partitioned according to their (non-)functional requirements. Services with different functional requirements were considered as service tasks, each of which included a set of services with identical but different non-functional requirements in terms of service quality and other properties. Any of these sets was a service group associated with a service task. In order to registry services, they provided a hybrid storage architecture in which service tasks were stored in one array because they were limited in number; and they used counting Bloom filter to store any of service groups related to these service tasks because of their numerosness. It's search algorithm has 2 phases: in the first phase, it searches service tasks within arrays based on functional requirements and, if any, in the second one, it searches respective counting Bloom filter based on requested non-functional requirements.

In [4], a service discovery protocol has been introduced for ad hoc networks on the basis of using attenuated Bloom filter. Their model was focused on discovering services within local domains where a large number of nodes are connected to each other on the basis of a wireless technology. Nodes declare their services via local dissemination messages. All nodes keep information about services obtained from their direct neighbors within attenuated Bloom filter. In this model, periodic dissemination is used to keep information update, deleting services which have not been declared for several periods.

III. AN OVERVIEW OF THE DATA STRUCTURE OF BLOOM FILTER

Bloom filter represents a compact brief data structure to store and display a set or list of elements, being capable of reducing space utilization for existence of some false positive [5]. This structure works mainly based on hash functions and can be used to code information in cases where a huge volume of data and information needs to be sent.

III.1. Standard Bloom filter

Structure of standard Bloom filter contains an m-bit array all cells of which are zero initially.

Adding elements of set $S = \{ x_1,...,x_n \} \quad (n \text{ elements})$ is done as follows [5]:

$X_i$ is added for each element while using K independent hash functions $h_{1},...h_{k}$ with range $\{1,...,m\}$, so These hash functions map each item in the universe to a random number uniform over the range $\{1,...,m\}$. Next, bits of $h_i(X_i)$ are set at 1 for $j=1,2,...,K$. An approach similar to elements addition is used to examine whether a specific element exists in Bloom filter. K hash functions are calculated for element $y$. If at least one of $h_i(y)$ bits is equal to 0 for $i=1,...,K$, this element does not exist in the set certainly. But if all bits equal 1, then this element exists in the set probably. Although
it's slightly probable that tested bits be set at value of 1, there will be a false positive. The probability of false positive of Bloom filter is expressed as follows:

$$P = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m} \tag{1}$$

that is, it's probable that still one of m bits of Array assumes value of 0 after adding n elements, therefore, probability of false positive means that all k bits being tested equal 1 as:

$$f = (1 - p)^k = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx (1 - e^{-kn/m})^k \tag{2}$$

On the basis of above equation, if factors are chosen as $K = \frac{m}{n} \ln 2$, false positive assumes the least value [5]. Standard Bloom filter does not support deleting elements from a given set, leading to false negative occurrence. To avoid this problem, counting Bloom filter has been provided.

### III.2. Counting Bloom filter

Counting Bloom filter is an array m-sized in which each locus is a counter instead of a bit. As an element is added, counters corresponding to k hashing increase by one unit, but they decrease by one unit when an element is deleted. In this structure, membership test is the same as that in standard Bloom filter. This method is free of any false negative. It is essential to select counters that are big enough to avoid counter overflow [5]. In proposed method, counting Bloom filter with 1-byte sized counters was used.

### IV. AN OVERVIEW OF PROPOSED METHOD

In this method, we used a hierarchical 2-level structure to register as well as inquire resources, in which nodes were classified into 3 categories:

- resource nodes, user nodes, and local grid information service nodes at top of which, in addition, a node lies as Grid Information Service(GIS). Local grid information services and GIS are organized at the first and second levels, respectively, both of which use Counting Bloom filter in order to store resource information. In this structure, as shown in figure 1, each resource node records its information at the 1st level in respective local grid information service.

![Figure 1. Topology used](image)
Counting Bloom filter, if there is any, it sends to users a list of discovered resources, otherwise it sends the request to GIS. GIS also searches requested resources in its own Counting Bloom filter, if there is any requested resource, it sends a global list of discovered ones to local information service, otherwise it sends to it the message of the lack of requested resources. Local information service sends the response to users and caches list of global discovered resources in its own counting Bloom filter if there is any one. Later the details of this method are outlined extensively.

IV.1. Architecture of storing resource information on the basis of Counting Bloom filter.

As mentioned in introduction of Bloom filter, factors are selected in the form of \( K = \frac{m}{n} \ln 2 \) in order to minimize false positive, where \( m \) is size of Bloom filter, \( n \) represents the number of resources, and \( k \) is the number of hash functions. CRC16 was used to define hash functions. Each of local grid information services has a Counting Bloom filter to store local resources. GIS also uses a Counting Bloom filter to store global resources. For simplicity purpose, all Bloom filters were of the same sizes. hash functions are used only on the sides of resource and user nodes. On resource side, position related to information about resources themselves are obtained using \( k \) hash functions and sent to local information service along with resource identifiers to be stored. In addition to storing resources on the basis of received positions, local information service sends positions and identifiers of resources to GIS to be saved via its own Counting Bloom filter. On user side, positions related to information of resources requested by users are obtained by using \( k \) hash functions and sent to local grid information service for searching purposes. Therefore, grid information services do not use hash functions directly. Figure 2 shows architecture of storing resources based on Counting Bloom filter. In this architecture, each position in Counting Bloom filter is connected to a list for storing resource identifiers. On basis of \( k \) positions obtained from hashing the resource information which is sent by resource to it's local grid information service, initially, value of counter in each of these positions increases by one unit in Counting Bloom filter within respective local grid information service. Then these positions are transformed into a string code in order from positions 1st to \( k \)th in this way that it considers each one as a string, connecting them to each other by "_", and obtaining respective codes to store resource id in the list connected to \( k \)th position. with this technique, similar resource identifiers are stored in one position on the list connected to the last position obtained from hashing the resource information. As seen from figure 2, for example, resource with information \( e \) is mapped to positions 2,7…4,12 in order, with 2,7…12 being position 1th to \( k \)th here. For these positions, string code obtained is "2_7_4_12". this code and identifier related to the resource are stored in the list connected to the last position obtained from hashing resource information, that is, position \( h_0(e) \) from Counting Bloom filter, which is equal to 4 in this example. As is observed, since information on all similar resources is hashing into the same position, their identifiers are stored on one list at one position of it. In figure 2, resources with identifiers \( id_1,..,id_4 \) are similar. With this method, similar resources are discovered more quickly.

IV.2. Process of resource registration in grid information service

In order for a resource in grid environment to be used, it must announce information about it's service. In this method, because Bloom filter is used, working on the basis of hash function, and since such functions act on the basis of strings, initially, each resource changes information on it's service into the form of a string, then, positions associated with the resource in Bloom filter are obtained by applying hash functions. As algorithm 1 indicates, these positions are sent Compactly along with resource identifiers to local grid information service in order to register resources at the 1st level, namely in Bloom filter of local information service, and to reduce bandwidth used.

With the trend described in previous section extensively and shown in algorithm 2, local information service stores it in it's own counting Bloom filter, next, sends positions compactly along with resource identifiers to GIS in order to register resource information at the 2nd level, namely GIS.

With a trend completely similar to that for registering resources in local information service, GIS also registers resource in its own counting Bloom filter.

Algorithm 1. Pseudocode of sending resource information to local grid information service by the resource node.

```
str=String(ResourceCharacteristics);
for i=0 to k-1 do
   bloom_index[i]=hash(str);
end for
index = compress(bloom_index);
send (index , Resource_id) to related RegionalGIS;
```
Algorithm 2. Pseudocode of registering resource information in grid information service

**Receive** (index, Resource_id) **from** Resource;

bloom_index = **extract**(index);

str = null; j = 0;

for i = 0 to k - 1 do
    j = bloom_index[i];
    bloomfilter[j].counter++;
    if i < k - 1 then str = str + j;
end for

bloomfilter[j].list.add(str, Resource_id);

**send** (index, Resource_id) **to** GIS;

---

**Figure 2.** Architecture of storing resource information based on Counting Bloom filter

**IV.3. Trend of resource request by users**

Users change information on resources they request into a string giving their own needs, then, they obtain positions associated with requested resources in Bloom filter by applying hash functions. Next, they send these positions along with their own as well as the task’s identifiers to local grid information service. In order to reduce consumption of memory, hence of bandwidth, obtained positions are compacted in the form of 2 positions in 1 unit while compacting user’s identifier and task’s identifier in 1 unit, then, they are sent to local grid information service. This trend is shown by algorithm 3.

Algorithm 3. Pseudocode of sending requests for resource from local grid information service by users

str = **String**(Requested_ResourceCharacteristics);

for i = 0 to k - 1 do
    bloom_index[i] = hash(str);
end for

index = **compress**(bloom_index);

id = **compress**(user_id, job_id);

**send** (index, id) **to** related RegionalGIS;
IV.4. Trends of resources inquiry and discovery

As shown by algorithm 4; upon receiving requests from users, local grid information service extracts positions from them. On this basis, positions in counting Bloom filter themselves; perform the trend of resources inquiry as follows: if counter value is 0 at least in one position in Bloom filter, requested resources do not exist in local grid. If counter values of none of these positions in Bloom filter is 0, then string codes pertaining to positions 1st to K-1th are created as stated previously and the list connected to position Kth in Bloom filter is searched according to string codes. If targeted member is found on the list, then the list of identifiers of similar resources registered in this member is sent along with task’s identifier to user. If the list is empty and/or is not but does not contain targeted member, then false positive are detected, indicating that no requested resource exists in local grid. Therefore, as a false positive appears in this method, it is detected and it has no effect on the trend, reflecting lack of resources, and in general, false positive decrease to 0. In the case that there is no local resource, the requests, in the same form received from user, are sent along with their identifiers to GIS to inquire global resources, with a similar trend, GIS searches its own Counting Bloom filter for requested resources. If it finds targeted and/or similar resources, it sends their list along with user’s request to local grid information services, otherwise it declares lack of global resources by sending user’s and task’s identifiers to local grid information services, which upon receiving list of global discovered resources from GIS, sends it along with task’s identifier to user and caches this list in its own Bloom filter according to positions. If it receives a message from GIS that there is no global resource, it informs user of the lack of requested resource in grid environment by sending task’s identifier. This trend is shown by algorithm 5.

V. EVALUATION OF PROPOSED METHOD EFFICIENCY

To examine proposed method efficiency, we simulated search and discovery of resources requested by users in 2 modes:

Algorithm 4. Pseudocode of resource searching in local grid information service

Receive (index , id) from user;
bloom_index= extract(index); j=0; str=null;
(user_id , job_id) = extract(id);
for i=0 to k-1 do
    j=bloom_index[i];
    if bloomfilter[j].counter==0 then
        send (index , id, RegionalGIS_id) to GIS;
        return;
    else if i<k-1 then
        str=str+'j';
    end if
end for
LocalResourcelist=search(bloomfilter[j].list , str);
If LocalResourcelist ==null then  // false positive
    send (index , id, RegionalGIS_id) to GIS;
    return;
else
    send (LocalResourcelist, job_id) to user;

Algorithm 5. Pseudocode of sending response to the search of global resources received from GIS to users by local grid information service.

Presence of global resources corresponding to the requested resources:
Receive (GlobalResourcelist , index, id) from GIS;
bloom_index= extract(index); j=0;
(user_id , job_id) = extract(id);
Cache (GlobalResourcelist) to bloomfilter based on bloom_index;
Send (GlobalResourcelist, job_id) to user based on user_id;

Absence of global resources corresponding to the requested resources:
Receive (id) from GIS;
(user_id , job_id) = extract(id);
send (job_id) to user based on user_id;
with using Bloom filter, and without it. GridSim 5.2 was used to do simulations. In method without Bloom filter, the list was used to register resource characteristics in local grid information services and in GLS. In both modes, we measured mean time spent on searching and discovering users requested resources, which is done by grid information services, as well as the number of bytes transferred over the network for each resource request from and for sending each response to users by placing an observer in local grid information services. Variations of the mean of spent times, obtained from simulations done for both modes, with increasing the number of resources and increasing the number of requests are shown graphically in figures 3 and 4, respectively. Also figures 5 and 6 illustrate changes in the number of bytes transferred over the network for each resource request and sending response to users in both modes with increasing numbers of resources and of requests, respected. In this simulation, users and resources are assigned to local grid information services randomly on the basis of structure shown in figure 1. Also characteristics of resource services and of resources requested by users are random. As shown graphically in figure 3, mean time spent on searching and discovering resources in grid information services with proposed method is much less than that of method without Bloom filter, being done with higher speed, since the probability of presence of local resources increases as the number of resources increases, for both modes, this mean time decreases with an increases in the number of resources. Figure 4 shows that overhead of grid information services increases with an increase in the number of requests so as mean time spent on searching and discovering resources with Bloom filter mode is much less than that without it, this mean is approximately fixed, with very little fluctuation, for both modes. As stated in previous section, in proposed method, for each resource request, a user obtains positions that need to be tested in Counting Bloom filter of local information service in order to search resources by hashing characteristics of requested resources, then, sends them along with user’s and task’s identifiers to local grid information service, in which case he/she needs not to send resource characteristics, therefore, compared to a normal mode, much less bytes are required to send requests and less bandwidth is used. In addition, since the largest positions obtained from hashing characteristics of requested resource is at most of the same size as Bloom filter and the size of Counting Bloom filter was set at maximally, each positions obtained, that is, value of \( hi(e) \) can have maximal \( \log(m) \) bits, therefore, in proposed method, in order to reduce the number of bytes transferred, in side user positions obtained be compacted in the form of 2 positions in 1 unit as well as user’s and task’s identifiers in another one and then user sends them.

In method using no Bloom filter for each request, users send characteristics of requested resources along with user’s and task’s identifiers to local information service. In this case much more bytes are used for requests compared to proposed method. For local grid information services, if there are local resources, identifiers of similar resources found are sent along task’s identifiers to users. For both methods, sending responses to users from local grid information service is done in the same way, with identical number of bytes being sent to users. In order to find global resources, if local resources are absent, local grid information service sends user’s requests, in the same form it received them, along with its own identifier to GIS, in which case the number of bytes sent to GIS from local grid information service is much less in proposed method than in one without Bloom filter. In the case that GIS find global resources, it sends the list of them along requests it received to local grid information service. So in this case, the number of bytes sent to local grid information service from GIS is much less in proposed method compared to that in method without Bloom filter.

![Figure 3](image-url)  
**Figure 3.** Diagram of variation of mean time spent on searching and discovering resources in grid information services with increasing number of resources for both modes using and not using Bloom filters.

Given the requests, local information service sends the list of global resources along with task’s identifiers to respective users and caches global resources list received in its own Bloom filter. Here again the numbers of bytes sent to users are the same for both methods. In the case no global resources commensurate with requests are found, GIS declares the absence of global resources by sending user’s and task’s identifiers to local grid information service. In this case, due to compacting user’s and task’s identifiers into unit, the number of bytes sent to local information service is, for proposed method, 4 bytes less than that for method without Bloom filter. But for announcement of absence of resources in grid, local grid information service sends the same number of bytes to users while sending task’s identifiers to them in both methods. In this simulation, we measured sum of these bytes for each request and, then, obtained mean of them for different number of resources as well as sum of them for different numbers of requests. Figure 5 shows that mean number of bytes transferred for each resource request is much less in method with Bloom filter than that is one without it, and as resources increase, the number of bytes used to send their identifiers to users increases in both methods because the number of similar resources may increase. Figure 6 shown that as the number of requests increases, the number of bytes transferred over network increases in both methods, but it is much less in the case using Bloom filter than that in one not using it, therefore, utilization of bandwidth increases with an increase in the number of requests, except that in the mode with Bloom filter, much less bandwidth is used than that in the mode without it given the number of bytes transferred.

![Diagram of variation of mean time spent on searching and discovering resources in grid information services with an increase in the number of requests for both with and without Bloom filter modes.](image)

**Figure 4.** Diagram of variation of mean time spent on searching and discovering resources in grid information services with an increase in the number of requests for both with and without Bloom filter modes.

![Diagram of variation of mean number of bytes transferred over the network for each resource request and sending responses to users with an increase in the number of resources in both method with/without Bloom filter.](image)

**Figure 5.** Diagram of variation of mean number of bytes transferred over the network for each resource request and sending responses to users with an increase in the number of resources in both method with/without Bloom filter.
VI. CONCLUSIONS

In this paper, we utilized the Bloom filter to discover resources in Grid information services. Results show that in proposed method, mean time spent on searching and discovering requested resources and the number of bytes transferred over the network for requests resources needs by users and send response to them are almost 80% and 78% less than those in the mode not using Bloom filter, respectively. It can be said, therefore, that Bloom filter represents a way for summarizing information on present resources and on requested ones and at times when a large number of resources is available and huge volume of information needs to be stored and when the number of requests increase in the network, making use of the data structure of Bloom filter along with an appropriate technique can be highly useful for optimizing the trend of resource registration, requested, search and discovery.

Using this simple structure saves bandwidth, which is an important resource for the networks, and minimizes the time spent on searching requested resources. By using storage architecture provided, not only there is no false negative in the search for requested resources in the counting Bloom filter but also the positive one is detected as soon as it occurs having no effect on the trend provided and, in general, it can be argued that there exists no false positive. Therefore, in this method, disadvantages of Bloom filter have been removed and its numerous advantages are exploited.

REFERENCES


