Site Selection in Swift Workflow Execution

Introduction

Basically, site will be selected dynamically in Swift. Site selection (scheduling) algorithm is an adaptive one considering previous jobs execution performance of sites. Based on my understanding of current Swift implementation, this document describes a set of complete site selection methods. The main difference from the current implementation is that, sites will be divided into different groups explicitly based on their status. Each time a target site will be only selected from one of group (called candidate sites set) based on their scores, which is promising to decrease the possibility of selecting bad performance sites. In addition, scores of each site will be adjusted according to its job execution performance such as submission time, queue time, job finishing status and so on.

Different Site Status

Divide sites into different sets:

1. Candidate sites set: these sites are candidates to be selected based on weighted-random selection algorithm. At the beginning, it includes all sites in sites.file with initial scores (***initialScore***).
2. Hibernated sites set: these sites comprise a queue according to in an arrival order. These sites couldn't be chosen for submitting jobs, but they can enter candidate sites when meeting some requirements. In addition, when they become candidate sites again, their initial scores would be reset to 0 or 1, which might mean new chance being evaluated again. At the beginning, this set could be null. However, it could include those sites, which are filtered when generating sites file based on calibration scripts. For large workflow, the initially filtered sites might become available again due to dynamic site performance fluctuation.
3. Overloaded sites set: these sites are already up to the limit (***max_jobs_per_site***) of maximum concurrent jobs allowed on them.

Switching Conditions of Site Status

When some conditions are met, the status of sites could be changed correspondingly.

1. Candidate sites set → Hibernate sites set:
    a. Job execution failure: when a site fails to finish a job (referred as ***temp failed site***), it should be frozen temporarily. A job execution failure might indicate that there is something wrong with this site and it is not proper to execution Swift job for this moment. Note that there probably are other jobs, which have been already submitted, on the ***temp failed site***, so those jobs should be allowed to continue.
    According to this, there are some kinds of situations:
        a1: there is no other job on this ***temp failed site***.
        a2: there are some other jobs on this ***temp failed site***..

Anyway, a temp failure doesn't necessarily mean the site couldn't be chosen forever. So these sites could re-enter candidate sites set.

b. Long queue time: when a job waits on certain site for too long time, that site would be hibernated temporarily. The key factor is the definition of "long time", and this should be a threshold time (***queueTimeThrottle***). Initially, this threshold could be set based on the queue time distribution of my globus jobs, which could be turned into CDF graph.

This threshold should be set carefully, avoiding too many sites being considered unavailable. In implementation, it can be also a parameter in properties file.

As far as implementation is concerned, there are two situations:

Replication-enabled: when replication is enabled, a replica job will be submitted to another site after a replication threshold time (note that it is different from above threshold time). Once one of replica gets started to execute, all other replicas would be cancelled. At this time, those sites with extremely long queue times could be judged and assessed as candidate sites or hibernated sites.

Replication-disabled: once a site finishes a job successfully, the queue time would be calculated and compared with the threshold time, deciding if it is still a candidate site or should be hibernated.

2. Hibernate sites set → Candidate sites set:

a. Candidate sites set are null: when all sites are either hibernated or overloaded, job could not be submitted any more. At this time, the head site of hibernate sites queue can be chosen to become a candidate site, so the site is entitled a new chance after an amount of time. For these sites, the score should be set to 0 when they become candidate sites again.

b. The ***temp failed site*** in hibernate sites queue finishes a job successfully: this corresponds to above situation **a2**. When that job is set as hibernated due to job execution failure at that time, there are other jobs on it. If the site can finish other job successfully, it indicates that that the previous execution failure is caused by a temporary error and this site is still not too bad. For these sites, the score should be set to 1 (or a reasonable number more than 0) when they become candidate sites again. This is a bit different from the first situation.

3. Candidate sites set → Overloaded sites set:

The number of job submitted to a site reach the limit (***max_jobs_per_site***) of allowed jobs on that site, the site would be considered as overloaded and no more jobs could be submit to it at this time.

4. Overloaded sites set → Candidate sites set

When the overloaded site finishes a job successfully, the number of jobs on that site would be definitely less than its limit (***max_jobs_per_site***). Consequently, the site will become a qualified candidate site again.

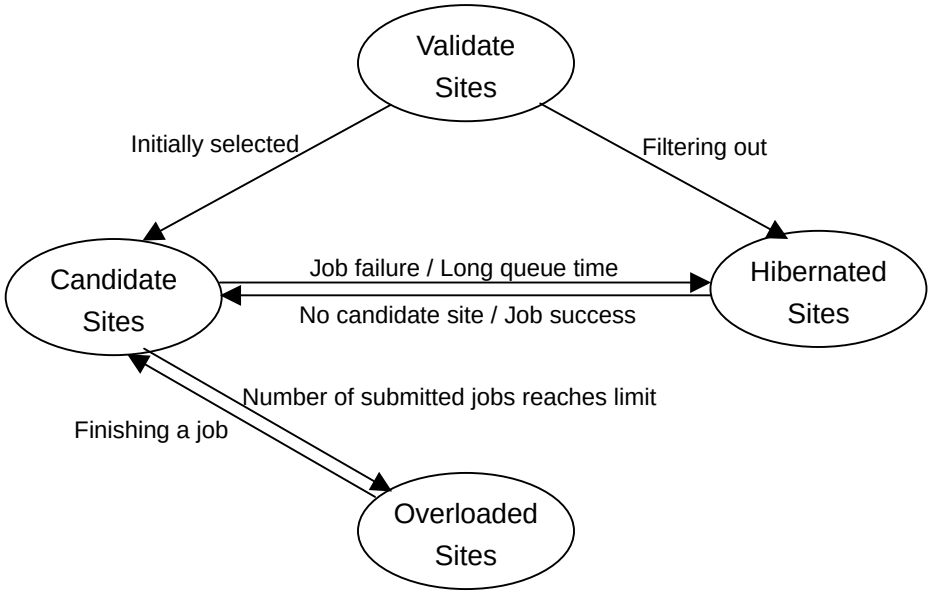

Fig. Sites Status Switching

Scoring mechanism

1. Increasing scores of sites
    a. A job is submitted to a site successfully (***jobSubmissionTaskLoadFactor***).
    b. A file operation is done successfully on a site (***fileOperationTaskLoadFactor***).
    c. A file is transferred to a site successfully (***transferTaskLoadFactor***).
    d. A site finishes a job successfully (***successFactor***).
2. Decreasing scores of sites
    a. A job is going to be submitted to a site (***jobSubmissionTaskLoadFactor***).
    b. A file operation is going to be done on a site (***fileOperationTaskLoadFactor***).
    c. A file is going to be transferred to a site (***transferTaskLoadFactor***).
    d. A site fails to execute a job (In fact, the site would be hibernated, so the score will be decreased to 0) (***failureFactor***).
    e. With replication enabled, once one of replica job get started to execute on certain site (referred as ***active site***), all of other replica jobs would be cancelled, so the corresponding sites' scores should be decreased at the meantime (***queueTimeFactor***). However it is worth noting that replications are not launched at the same time. So it's unfair to decrease scores of all other sites. Only those sites whose queue times are longer than that of ***active site*** should be penalized.
    f. When the job submission time of a site hits certain threshold, the score of that site should be decreased (***submissionTimeDelta***).

Strategies

1. Replication

After a job is submitted to a site, the queue time will be calculated and recorded. If the queue time

is up to a threshold (***replication.min.queue.time***), a replica job will be launched. Once one of replica job begins to execute, all of other replica jobs would be cancelled. The max number of replica jobs is defined by ***replication.limit***. Replication can provide more guarantees for a single job to get started as soon as possible, but it will incur overhead at the meantime. Because Swift uses GRAM to submit jobs, and the submission stage in GRAM is one of the most CPU expensive stages. In addition, ***replication.min.queue.time*** could be set based on the knowledge of queue time distribution of target Grid environment.

Site Selection Algorithm

1. weighted-random selection: randomly selecting a site from candidate sites set, the higher score a site has, more probably the site would be selected.
2. best scores selection: always select the site with highest score from candidate sites set.

Note: currently weighted-random selection algorithm is adopted in Swift. Best scores selection method might be also tried. Comparisons could be made between these two methods.

Some Controlling and Tuning Parameters

1. ***Score***

This is the most important parameter for site selection. It is the reflection of comprehensive performance of each site. The final site selection decision is made based on this parameter. Currently, I use some simple methods to set an initialScore for each site involving all my observation sites set based on the results of some calibration scripts. It is a kind of exploration. There are two functions of setting initialScore. One is to filter some sites with extremely bad performance in real time; the other one is to help scheduler to make proper decision at the beginning. From experiment to real production, some changes are requirements.

Solution 1: Improving my current calibration scripts.

Save those results in database, instead of separate files, which could support all kinds of queries. In my experience, I feel that there is always some changes in OSG and such fluctuations should be allowed due to the nature of Grid environment. Consequently, static methods of probing remote sites are not very suitable. In my experiments, I often try to discover resources actively and manually using some monitoring systems such as VORS in OSG. So practical dynamic resource discovery for Swift, which could be combined with application deployment, might be a topic for exploration.

Solution 2: Utilizing some current Grid services.

For example, ReSS in OSG. The Resource Selection Service (ReSS) is the information provider which collects data from each supporting sites and publishes such data in Condor ClassAd format. In addition, OSG Matchmaking (OSGMM), which in fact is a Condor-G job, would use such information to do matching. The information provided by ReSS might be transformed and utilized by Swift scheduler. However I'm not sure if OSGMM could replace the site selector in Swift, because it requires a detailed job requirement description to make matching. There is no such a job description. Maybe for Swift users, they just want their workflows to be executed and might not be willing to write such descriptions. In order to use

OSGMM, Swift has to generate such a job description according to user applications.

2. ***TScore***

   It is derived from Score and it is a function of bounding the value of score.

$$TScore = e^{B*\log(scoreHighCap)/\pi*\arctan(C*Score)}$$

3. ***jobThrottle***

   This parameter is a controlling throttle which will affect the maximum number of allowed jobs submitted to a site.

4. ***max_jobs_per_site***

   The maximum number of allowed jobs is calculated using the formula:

   max_jobs_per_site = 1 + ***jobThrottle*********TScore***

5. ***queueTimeThrottle***

   This parameter is a time threshold deciding if a site where a job is waiting for execution too long should be thrown into hibernation status. It could be a variable which is given a initial value according to experience. During execution, this threshold could be adjusted dynamically based on previous job executions. The simplest way to set this value is as follows:or example,

   queueTimeThrottle = *average queue time* + c

   Note: c is modifying factor, and it can be a constant set in swift.properties.

6. ***jobSubmissionTaskLoadFactor*** / ***fileOperationTaskLoadFactor*** / ***transferTaskLoadFactor*** / ***successFactor*** / ***failureFactor***

   These parameters are factors used to adjust scores of sites. They should be variables, because chances are less finding best value combinations of these parameters for all applications. However, for specific application and grid environment, optimal values could be found.

7. ***queueTimeFactor***

   With replication enabled, this parameter could only be used to change the scores of sites.

8. ***submissionTimeDelta***

   Currently, it is calculated with the following formulas:

$$submissionTimeDelta = submissionTimeBias + submissionTimeFactor * submissionTime / 1000$$
$$submissionTimeBias = -BASE\_SUBMISSION\_TIME * submissionTimeFactor$$
$$submissionTimeFactor = -successFactor / (maxSubmissionTime - BASE\_SUBMISSION\_TIME)$$

9. ***replication.min.queue.time***, ***replication.limit***

   These parameters together control the number and producing frequency of replica jobs. Like ***queueTimeThrottle***, this parameter is also a threshold time. Its value could be derived from the learning of queue time distribution in target Grid environment such as OSG and Teragrid.