Graph Based File Prediction

Why use graph

Previous work use only the time correlation, i.e. access history, to predict the future file access. While we use graph based algorithm trying to exploit the relations between files.

A major assumption made in graph based algorithm is that: **if a file had been accessed a lot with popular files, it is likely that this file will become popular if it is not popular already.**

Introduction

To exploit relations between different files, we build a graph.

Basics of the graph

- Nodes:
  Each file is represented as a node in the relation graph. Nodes have a property called placement, which could either be SSD or HDD.
- Edges:
  When two files have been accessed together, then there will be an edge between two nodes. By together, we mean in a short time period, which we call edge adding window. Edges will expire after a certain time period, which we call edge removal window.
- Weights:
  Edges have weights, that is, two nodes can be accessed together more than once, the weight of an edge represent how many time two nodes have been accessed together.
- Connections:
  The sum of weights of edges with all neighbors of a node. The more connections a node has, the file are related to more other files.

- Ratio of connections:
  The sum of weights of edges with neighbors in SSD / The sum of weights of edges with all neighbors (connections of a node).
  This is the main metric we are going to use in graph based algorithm.

Parameters of the algorithm:

- Edges adding window:
  When two file have access time that are both inside this window, an edge will be added between two nodes (files).
- Edges removing window:
  After how long an edge will be removed.
- Update period:
  How often do we update the file placement.

Trace file format:

There are three trace file I have done experiments on. Here I want to talk about the darshan trace under HPC environment.

I use the whole 2013 darshan traces. But you should know that this is trace does not include all the file access in the system.

Below is the original application based trace (long_trace-alcf-read.fo):

```
#<rank> <file> <counter> <value> <name suffix> <mount pt> <fs type>

0   2653436057101099323 CP_SIZE_READ_0_100  1     ...4188441115/intrepid-fs0gpfs

0   2653436057101099323 CP_SIZE_READ_1K_10K 1     ...4188441115/intrepid-fs0gpfs

0   2653436057101099323 CP_F_READ_START_TIMESTAMP  0.125132    ...4188441115/intrepid-fs0
    gpfs

0   2653436057101099323 CP_F_READ_END_TIMESTAMP    130.209102  ...4188441115/intrepid-fs0
    gpfs
```

So we have access histogram and access start and end timestamp for each access.

File name are hash values of original file name, and it is said to be consistent across traces.

Details regarding darshan trace please refer to it's document.

The algorithm

Overview:

The algorithm contains mainly three part:

- Preprocess trace file:
  Change the format of the trace that will be easier for main algorithm to use.
  Also, record the optimal placement for each day.
- Main algorithm:
  - Generate graph:
    Parse the trace file to store the relations of file accesses
  - Update placement:
    Based on the information derived from parsing the trace, generate a rank for each file, and update the placement using the rank

Preprocess trace file:

Based on the original darshan trace, combine the application based trace together and sort them in ascending time order.

For each file access, we know the start and end access time, we can also calculate the approximate access size from the histogram. We will divide the access period into several records, and divide access size equally to each record.

Preprocessed trace file format (preprocessed-alcf-3.txt):

```
# time, file name hash value, access size

1357000156,12849142095052265740,153600

1357000156,1280369988056642784,550

1357000157,9009777162045495984,5853050
```

By going through the trace file, we can also get the most popular files of each day, which can be used later in GOD mode.

Most popular files of each day (opt-alcf-3.txt):

```
# file name hash value, access size

6542989217802362368, 38500
```

```
------- 2 2763 6481 2763 5503 5269841318837 210184309817277 -------
2276924299421848606, 71883964714019
```

Note that the line with “------” mark, it is the summary of each update period, which is 3 day here.

There are 7 numbers in this line separated by space, there meanings are:

- update id:
  2 meaning this is the third update iteration (start with 0)
- number of new files in this update iteration:
  there are 2763 new file in this iteration
- number of all the files till this update iteration:
  there are total 6481 files
- number of new files being accessed:
  usually this is same as number of new files
- number of files being accessed:
  there are 5503 of the total 6481 files have been accessed in this iteration
- size of new files being accessed
- size of files being accessed

Generate graph:

The metric we are going to use for file prediction in graph based algorithm is ratio of connections of each file, during the implementation, I found that generating the graph first and then calculate this metric is very slow. So instead of generate the graph, we use the following method to calculate this metric.

We record the number of connections with neighbors in SSD and HDD respectively. To get such information, we keep track of how many access of the whole system are from SSD and HDD respectively in a most recent short time period, which is the edge adding window, with two variables. When a file has been accessed, we will use following step to update their record of connections with neighbors in SSD and HDD:

- We will first update the above two variables to make sure information older than an edge adding window to current time have been removed.

- Then we update the record of connections with neighbor in SSD and HDD of this file with this two variables.
- Finally we update the above two variables again, to include information of this file access record.

Although for each file, this method only count the connections of neighbors that has been accessed before it, we can still use it to get a good approximation since access from SSD and HDD in a short time period of the whole system is not changing dramatically.

There are two ways to expire edges:

- reset: reset the weight of the edge to 0, so that they will not effect the future calculation of ratio of connections
- fade: divide the weight of the edge with an fade factor, so that they will effect future calculation. but they will become less and less important as time goes by.

Update placement:

The system will update the placement for each update period. There are two mode: normal update mode and GOD mode for this part. Normal update mode is for us to evaluate our algorithm while GOD mode let us know what is the best we can do.

We will calculate a rank for each file to update the placement. The only difference of the two modes are the way they calculate the rank. For normal update mode, the rank is calculated based on the information we collected during current update period , while for GOD mode the rank is calculated based on the access of future.

We will put a certain percentage (based on SSD/HDD storage capacity) of files with highest rank into SSD, and the rest of files into HDD.

For normal update mode, we calculate rank for each file based on the following metrics:

- Ratio of connections (information of relations of file access)
- Access frequency in previous update period (information of file access increasing speed)

- Accumulated access frequency (information of access history)

For GOD mode, we solely calculate the rank based on access frequency of the future.

The evaluation

Benchmarks

To evaluate graph base algorithm, we use the following benchmarks:

- number of access from SSD of each update period
  the better placement we have, the higher this number we will get.
- Number of files being moved of each update period
  this is the overhead of the algorithm, a higher file movement means the algorithm has a higher cost.

Results

1. Normal update mode

By using the normal mode for update, the result shows that how the graph base algorithm works for darshan trace.

Fig.1 shows placement performance by using ratio of connections as main metric.

Fig.2 shows placement performance by using accumulated access frequency as main metric.

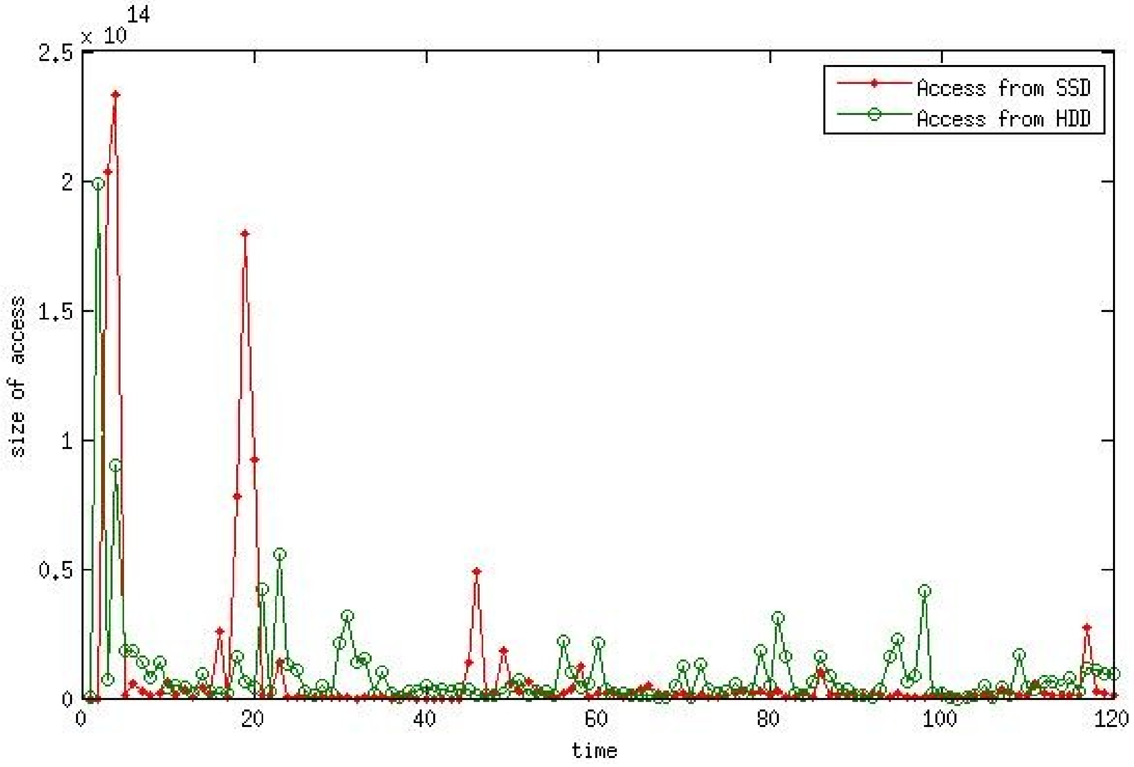

Figure 1: Access from SSD/HDD by using ratio of connection as main metric for prediction

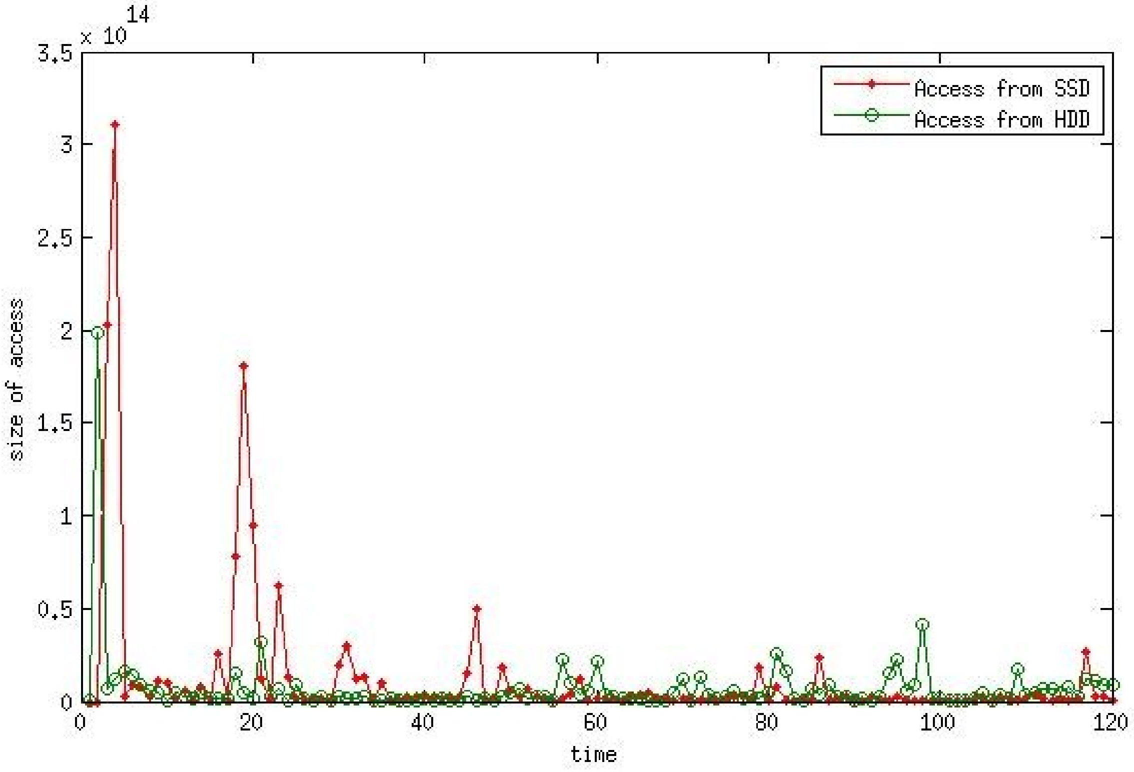

Figure 2: Access from SSD/HDD by using ratio of connection as main metric for prediction

2. GOD mode

By using GOD mode for update, we can know the best we can do.

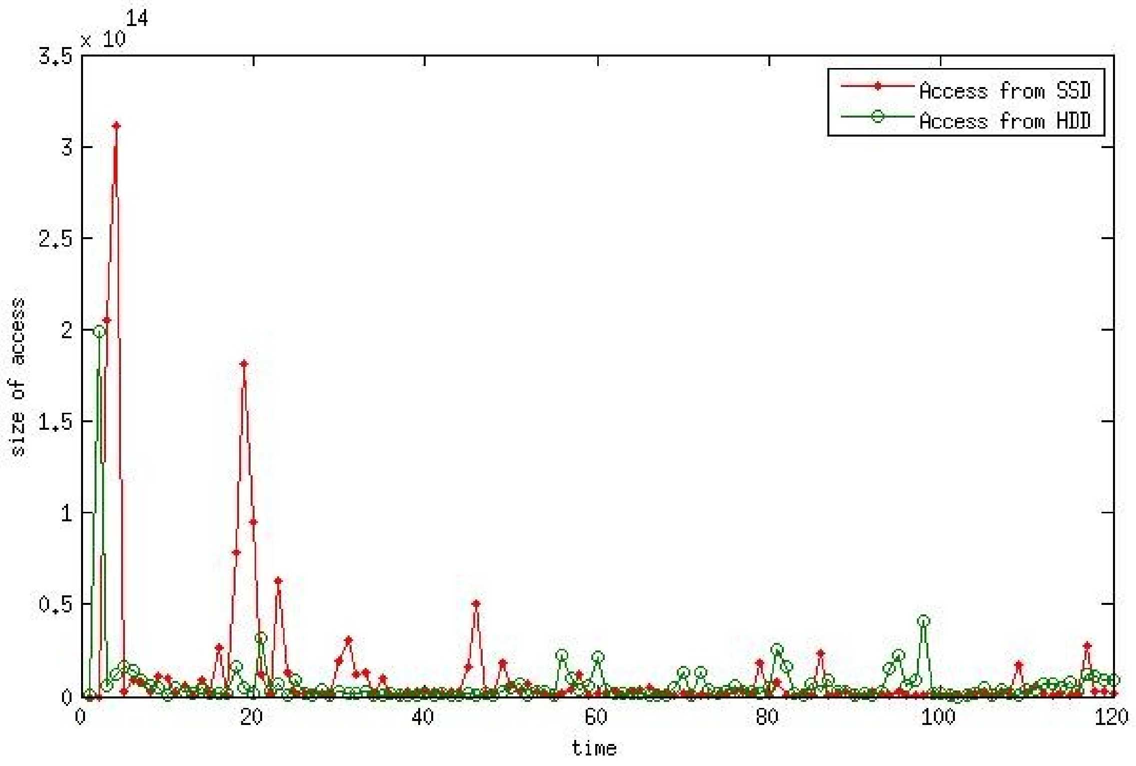

Figure 3: Access from SSD/HDD by using future access as main metric for prediction

3. Compare of the result of graph based algorithm with result of GOD mode

In Fig. 4, compared to ratio of connection, it seems that accumulated access frequency is a better metrics for placement.

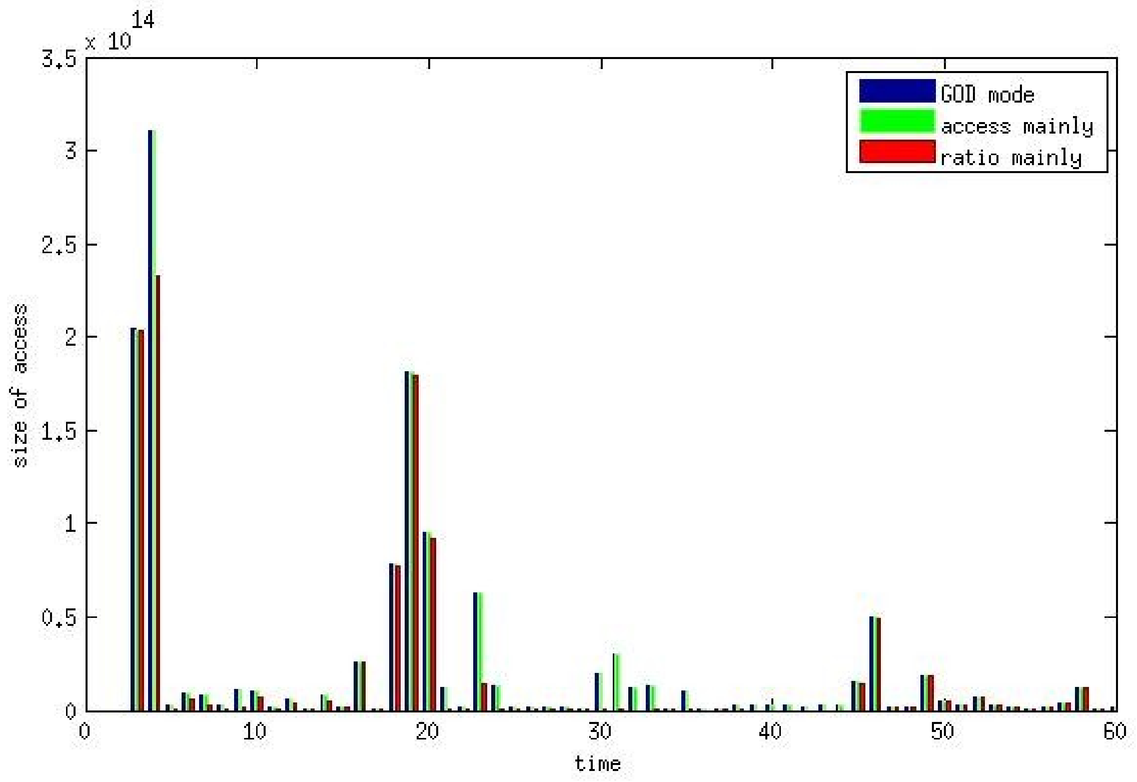

Figure 4: Comparison of access from SSD of three methods above (time are halved to see the figure more clearly, trends are the same for the rest half)

Extended Results

1. **Does the ratio of connection have potential for improving the placement? (results here are based on another trace, called LASR trace)**

Since the graph based algorithm does not work well with the trace

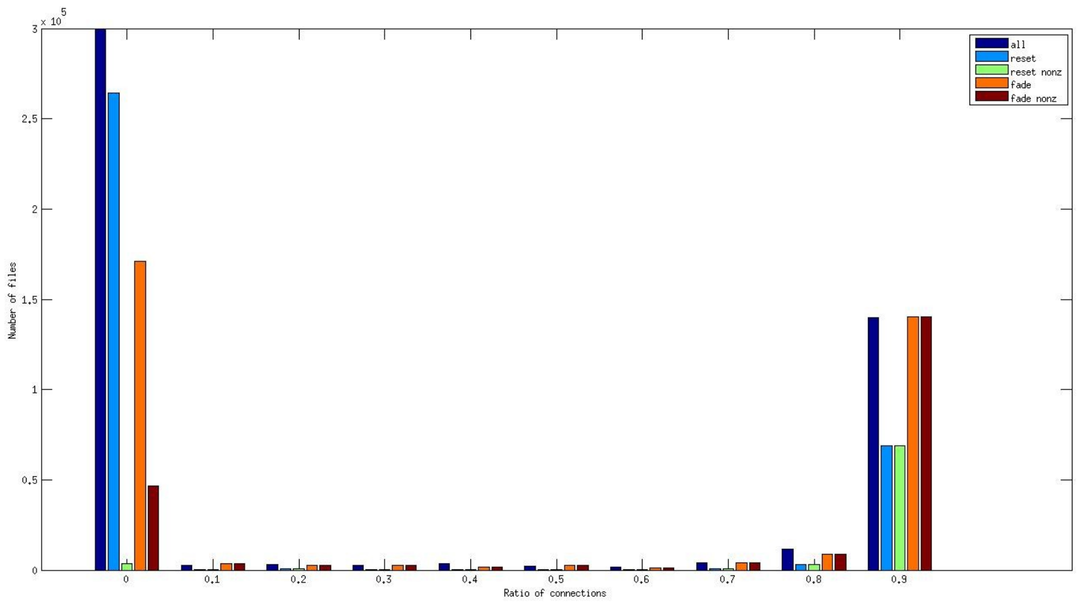

Figure 5: The ratio of connections of files with GOD mode

file, a more important question is whether or not this metric has the potential to provide useful information for the placement.

Fig. 5 shows the histogram of ratio of connections when we using GOD mode for the placement. There are five histogram in this figure:

- all:
  histogram of ratio of connections of all files, with edge expire strategy “reset”
- reset:
  histogram of ratio of connections of files in SSD, with edge expire strategy “reset”
- reset nonz:
  histogram of ratio of connections of files with non-zero connections in SSD, with edge expire strategy “reset”.
- fade:
  histogram of ratio of connections of files in SSD, with edge expire strategy “fade”
- fade nonz:
  histogram of ratio of connections of files with non-zero connections in SSD, with edge expire strategy “fade”.

From this result, we can see that the ratio of connections may help the placement of part of the files, but not all. When using the strategy “reset” to expire edges, a lot of nodes may have zero connections.

2. **How good is the performance of using only the accumulated access frequency? (results here are based on another trace, called LASR trace)**

If we calculate the area under curve “access from SSD”, we will have a number related to roughly how good this strategy works.

- GOD mode : 1.89E10
- using accumulated access frequency only : 1.76E10

There is not much room left for the performance improvement, simply using accumulated access frequency is good enough. Why?

Two possible reasons:

- There are several files that are constantly highly accessed.
- The access frequency difference between most accessed files and the other files are not very big. So there is no big difference if we put 10 most popular file or we put 10 less popular files.

3. **What might be the reason that even GOD mode doesn't work very well for the darshan trace?**

Even in GOD mode, we cannot put files into SSD before it is generated (first accessed). That is, even GOD mode cannot handle a trace with a lot of new files being created every update period. So maybe look at the life span distribution of darshan trace may give us some clue.

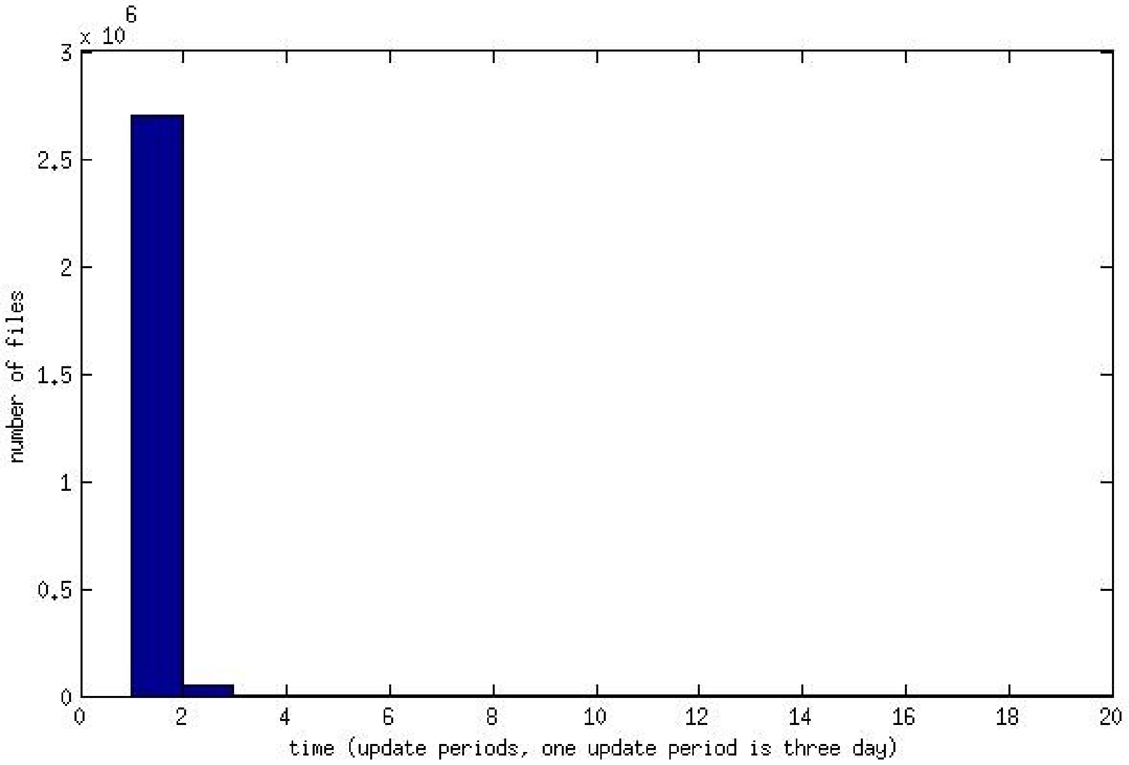

Figure 6: Histogram of life span of files in darshan trace

We can see from Fig. 6 that most of files have a life span shorter than three day, which means that a lot of files have been created every update period and they are only being used for a very short time period.