



OPTIMIZING SOLUTIONS FOR ATMOS



Krasimir Miloshev
Solutions Architect
EMC Corporation

EMC²

Table of Contents

Introduction 3
 EMC Atmos Terms and Definitions 3
 Defining the problem 5
A new optimizing approach for assigning nodes to tenants 5
 K-center of Graph 5
 Heuristic Algorithm for finding K-center of Graph 6
 Practical Solution 7
Conclusion 10
Bibliography 11
Appendix 11

Disclaimer: The views, processes, or methodologies published in this article are those of the author. They do not necessarily reflect EMC Corporation’s views, processes, or methodologies.

Introduction

EMC Atmos Terms and Definitions

Atmos[®] is a cloud storage platform that lets service providers store, manage, and protect globally distributed, unstructured content at scale. Atmos provides the essential building blocks to implement a private, public, or hybrid cloud storage environment. It's optimized to store, manage, and aggregate distributed big data across locations through a common, centralized management interface. Atmos delivers flexible access across a broad range of network topologies and access methods, from traditional applications such as web applications deployed on Windows, Unix, or Linux platforms to more modern multi-platform mobile devices. For added flexibility, Atmos supports legacy applications that rely on EMC Centera[®] SDK. The net result allows users and applications instant access to data, in a multi-tenant environment designed to deliver storage as a service.

Node - A physical server containing a collection of Atmos services. Each node contains one client, which can be a Web services client or a file system (NFS, CIFS, IFS) client.

Rack - A set of nodes on one physical rack.

Installation Segment - One or more racks, comprising a set of nodes that share the same "private," management subnet. Functionally, this is a set of nodes that share the same *master node* (the first node installed in each installation segment). The other nodes in the installation segment are slave nodes.

Resource Management Group (RMG) - A collection of installation segments that share a single IP domain. In almost all cases, this is equivalent to a subnet on the "public", customer network. Multiple RMGs can be created on the same subnet, as long as each RMG has a unique multicast address. RMGs are responsible for monitoring and discovering nodes within the subnet.

Location - Typically identifies the physical location of a set of RMGs. The RMG's location is specified during system installation.

Master node - The first node installed in each installation segment. It always has -001 appended to its node name. When a new RMG is added to the system, it has one installation segment, hence one master node. If more installation segments are added to that RMG later,

there are more master nodes (an RMG with N installation segments has N master nodes). The first master node in an Atmos system is the initial node.

Tenant - Stronger than simple access control, tenancy is a logical partitioning of data and resources. Each tenant appears to have unique and sole access to a subset of the system resources. Atmos administrators can create conceptual subsets of the storage resources within an Atmos system. Each subset is called a *tenant*, identified by a name that is unique. Atmos is architected to support multiple tenants or separate groups of resources. Those tenants:

1. Are identified by a unique name and namespace, which spans multiple locations.
2. Are isolated from other tenants on the system and managed by the tenant admin.

Tenants can:

1. Define unique policies so that you can assign the optimal policy definition for different groups or applications within those groups.
2. Have access nodes for Web Services and CIFS or NFS assigned for their private use.

Each Atmos tenant can be further subdivided into *subtenants*. Each subtenant has their own set of policies, users, and data access. When an Atmos tenant is created, the system automatically creates a default subtenant of the same name. Figure 1 shows an Atmos system with two RMGs: Boston and London. This system has two tenants (Tenant A and Tenant B) who are assigned different physical access nodes. Each of the tenants is further subdivided into two subtenants. These subtenants are also assigned different access nodes.

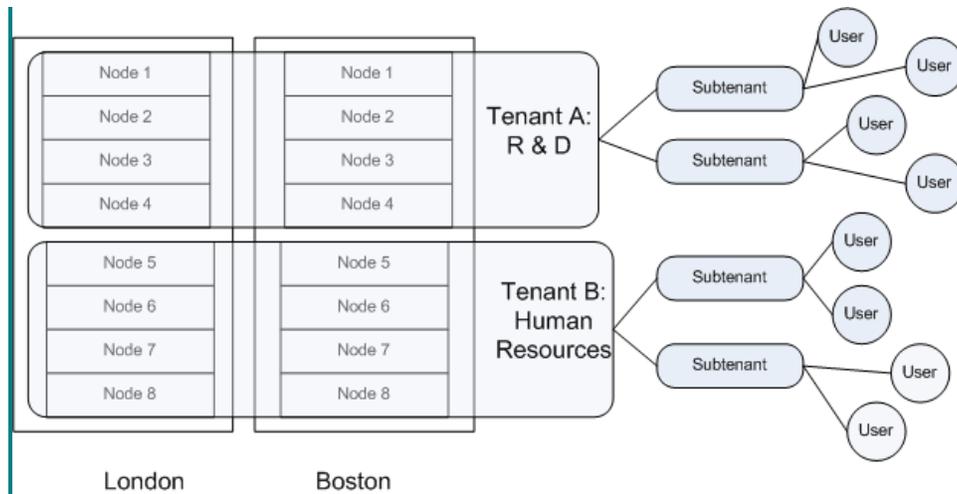


Figure 1

Defining the problem

Tenants are logical terms, but they are affiliated with specific physical resources – Atmos nodes and business locations where those tenants operate. These physical resources can span different geographic areas, between which might be different communication characteristics. For instance, tenants can use the cloud to store/back up data on a daily or weekly basis. Every time new tenants are added in Atmos, one or more nodes is assigned to them. The question is whether that whole assignment process is good enough, can it be improved, and eventually, what technique could be used to optimize that process. Practically speaking, the idea is to create a workload optimizing scheme to minimize overall data transfer time between customer locations and the cloud. To resolve that problem, a graph theory model will be used.

A new optimizing approach for assigning nodes to tenants

K-center of Graph

A common problem in Graph Theory is finding the p-center of graph [1]. The goal is to locate vertices on the graph, which represent locations, in order to minimize the cost. When $p=1$, then we have to 1-center algorithm is called Graph Center Algorithm [2]. Much more difficult to analyze is the situation when $p>1$, despite the fact that efficient heuristic algorithms have recently been developed for $p=2,3,4$.

Until recently, k-center problems were believed to be among the most difficult graph problems to solve. Algorithms that efficiently solve problems of considerable size (e.g., $n = 200$, $p= 5$) are so-called "relaxation algorithms [4]. Relaxation is a simple method used to optimally solve a large location problem by solving a sequence of small sub-problems. Fortunately, optimality can

be achieved even though each sub-problem need not be optimally solved. Relaxation algorithms are types of iterative algorithms. But practically speaking, while traditional iterative algorithms need to solve a few large problems, relaxation algorithms need to solve many small problems.

The idea is to solve the k-center problem by first solving, optimally, the 1-center, then the 2-center, and eventually the (k-1)-center problem. The motivation is to start solving the p-center problem when we a reasonably tight upper bound on the solution already exist. The experiments show that relaxation is particularly suited for problems with relatively small values of k (for instance $k < 10$). In our particular case, relaxation enables us to solve problems which were previously considered too large.

When it comes to relatively small values of k and n, optimum algorithms are preferable, since in the real world we have to deal with smaller values and have already found algorithms for resolving smaller cases. As p grows larger, (for instance, $k > 10$), the number and size of sub-problems does so as well and relaxation method may lose its advantage. Generally, when talking about finding a best-solution algorithm, we are acquainted mostly with so-called optimal algorithms; those that provide optimal solutions to problems. However, many optimal algorithms require so much time that they cannot be used in practice [3]. In real word scenarios, most problems have to be solved before a given deadline. Thus, people invented heuristic algorithms which intuitively seem to obtain optimum solutions sometimes and obtain pretty good and even near-to-optimum solutions in most cases. Most heuristic algorithms are based on human intuition and are therefore difficult to analyze.

Heuristic Algorithm for finding K-center of Graph

Our idea for finding k-center of graph based on heuristic approach is:

1. Dividing the array of vertices' values into separate k arrays, based on some existing load balancing algorithms such as FABP (fast algorithm for balanced partitioning). The idea behind this algorithm is a given set is divided into k balanced subsets using as many steps as k.
2. For each of the newly generated k arrays, we apply the 1-center algorithm to determine all of the k-centers.

A description of the algorithm for finding 1-center [2]:

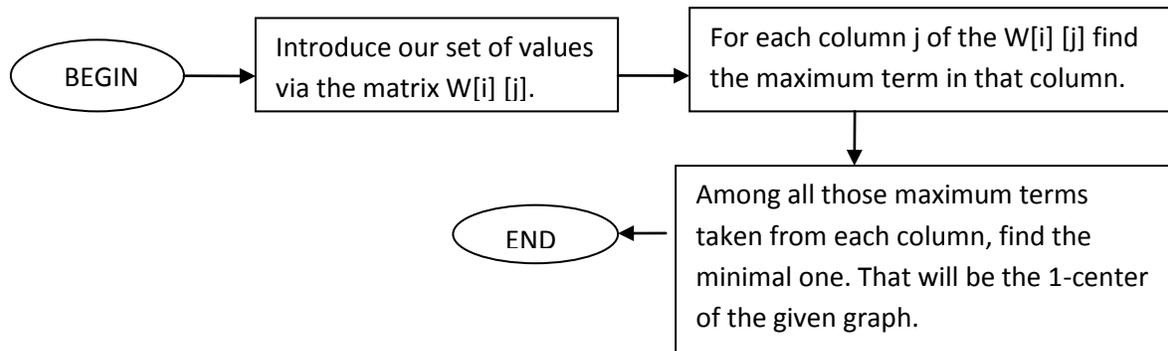


Figure 2

Practical Solution

The new heuristic method for finding k-center of graph is an original idea of the author of this article. For instance, consider a global company which is a candidate for using cloud and wants to use Atmos for backing up data on weekly basis. Assume $n=8$ customer physical locations where the customer's data centers are located. Those locations can span different geographic areas separated by hundreds and thousands of miles. Based on the Atmos node capacity information, the customer wants two Atmos nodes to be assigned to him as a tenant. Now, assume six different possible locations n_1, n_2, \dots, n_6 for the Atmos available nodes and let those nodes span different locations as well. Our goal is to find the "best" two locations among those six nodes to minimize transfer time between the customer/tenant locations and the Atmos nodes. The average weekly data amount (in GB) to be transferred/ backed up for each of those customer/tenants location is presented by the matrix $D(j)$ (Table 1). The average weekly amount of data are statistically collected over a long period of time.

1	2	3	4	5	6	7	8
3,400	4,200	4,800	3,200	3,900	3,660	4,198	2,900

Table 1

The data transfer rate between each of the customer/tenant locations and the available Atmos nodes are shown in Table 2. The average transfer rate (MB/sec) is statistically collected over a long period of time with different storage amounts.

	1	2	3	4	5	6	7	8
n1	21.25	13.76	10.21	10.62	11.64	16.50	17.71	12.09
n2	16.60	20.28	11.44	12.72	20.28	17.14	8.45	16.47
n3	19.51	12.03	29.09	12.63	19.59	30.76	30.37	17.26
n4	10.03	12.85	19.39	26.66	26.62	14.47	13.16	9.03
n5	9.72	15.23	10.68	9.35	27.27	12.70	22.15	12.78
n6	9.40	20.67	10.16	11.88	9.68	19.26	19.35	12.70

Table 2

Dividing the amount of each row in Table 1 by each of the values in Table 2 (divining two matrices) results in a new matrix, presented in Table 3. This matrix contains so-called projected average storage transfer time (in seconds) or average storage backup time for each of the tenant locations over each of the available storage nodes.

	1	2	3	4	5	6	7	8
n1	160	247	333	320	292	206	281	198
n2	253	207	367	330	207	245	497	255
n3	246	399	165	380	245	156	158	278
n4	319	249	392	120	108	221	243	321
n5	401	256	365	417	143	307	176	305
n6	389	177	360	308	378	190	189	288

Table 3

Now we have to find which two nodes among n1, n2, n3, n4, n5, and n6 are best located to minimize data transfer time between the tenant physical location and the nodes. The matrix W in our case will be the one represented in Table 3. We will apply the previously described heuristic approach by implementing the following 2 steps:

1. Dividing the array of Table 1 into two separate arrays, based on the previously developed FABP, and shown in the Appendix.
2. For each of those new arrays, we apply the 1-center algorithm, shown in the Appendix.

1	2	5	8
3,400	4,200	3,900	2,900

Table 4

Here we have divided our primary array from Table 1 into two new arrays after applying a load balancing distribution algorithm – see Table 3 and Table 4.

3	4	6	7
4,800	3,200	3,660	4,198

Table 5

The primary array was divided into two new arrays thus the difference between the sums of storage amounts for each of them is minimal. Practically, we have distributed the load over two new arrays so the total sums are either equal or have minimal differences. Let's now create two additional arrays (shown Table 6 and Table 7) based on projected time for storage backup from each of the tenant locations 1, 2,....8. We have to find the minimum term among all maximum column values. In Table 6 we have maximum values of 253, 256, 292, and 321. The minimum value is 253 at node n2.

	1	2	5	8
n1	160	247	292	198
n2	253	207	207	255
n5	246	256	143	321

Table 6

Again, we have to find the minimum value among all column values in Table 7, which are 392, 399, 221, and 243. The minimum value is 221 at node n4.

	3	4	6	7
n3	165	380	156	158
n4	392	207	221	243
n6	360	399	190	189

Table 7

Node n4 and node n2 are the 2-centers of the logical graph, and therefore, the “best” two nodes (among all 6) to be assigned to the new tenant from A time/performance perspective. This mechanism can be implemented as an additional optimization service built into the system and, when access nodes are assigned to tenants, they can decide whether or not to use this service.

Conclusion

In this case we are resolving the task to optimally assign nodes to new tenants. Our general task was to find the optimal nodes, among all available Atmos nodes. Our work is based on applying a heuristic algorithm for resolving the k-center problem where $n=8$ (number of possible locations) and $k=2$ (number of nodes to serve those locations), and where k is a subset of $p=8$ (number of all possible nodes). This is based on dividing the set of customer locations into $k=2$ separate balanced subsets, and then finding the 1-center of each of those $k=2$ subsets. Thus, we were able to find the heuristic k-center of the whole set. The new heuristic method for finding k-center of graph, based on dividing the whole set of vertices onto k balanced subsets and then finding 1-center of each subset, was developed by the author of this article. A software code for implementing those two separate steps of the suggested method—dividing a set into k separate balanced subsets and finding 1-center for each of the subsets—has been developed.

Bibliography

- [1] Hakimi S., 1978, On p-center in networks, *Transportation Science*, 12(1):1-15
- [2] Evans J., Minieka E., 1992, *Optimization Algorithms for Networks*, USA, Dekker Inc.
- [3] Hu T.C., Shing M.T., 2002, *Combinatorial Algorithms*, USA, Dover Publications, Inc.
- [4] of Handler G., Chen R. , “Relaxation method for the solution of minimax location-allocation problem”, 1987

Appendix

Our heuristic method with algorithm is based on combining two separate exact algorithms—FABP and 1-center. FABP is an algorithm for dividing a set of elements into separate balanced partitions, and 1-center is an algorithm for determining the centers of each of those partitions/subsets.

Program FABP (fast algorithm for balanced partitioning)

Short description

The major idea of this algorithm is:

1. Sorting the elements of the input array in decreasing order using any of the known sorting algorithms and determining the average value (pivot) of the input array's elements.
2. Algorithm has as many steps / passes as the number of partitions we want to divide the input array into. If we have to divide the input array into k balanced partitions, the number of steps to get those k partitions is exactly k.

Code implementation in C++

```
// FABP.cpp
#include <iostream.h>
#include <math.h>
#include <stdlib.h>
typedef int t_element;
const int nmax = 10000, key_min = 0, key_max =
2001, kmax=4, maxint=100000;
void bucket_sort( t_element a[], t_element pom[], int n, int kmin, int
kmax1 )
{
    int key, number, i;
    for( key = kmin; key <= kmax1; key++ ){
        pom[key] = 0;
    }
    for( i = 1; i <= n; i++){
        pom[a[i]] = pom[a[i]] + 1;
    }
    i = 1;
    for( key=kmax1; key>= kmin; key-- ){
        for( number=1; number<=pom[key]; number++ ){
            a[i] = key;
            i++;
        }
    }
}
```

```

    }
}
void main( void )
{int n=10000 , index,i,k,jpmin,jomin,j,jj;
double temp,piv,delta,dpmin,domin,s,snova,mmax,mmin,
server[kmax+1];
    t_element a[nmax], pom[key_max],used[nmax];
    while( n > 9999 ){
        cout << " Insert the number of elements of the array
(<1000): ";
        cin >> n;
    }

/*  for( index = 1; index <= n; index ++ ){
        while( a[index]<1 || a[index]>100 ){
            cout << " enter element (>0&<101) : " << index <<
" : ";
            cin >> a[index];
        }
    }*/
int r1,r=2000;
srand(r);
r1=rand();
for (i=0;i<=n;i++)a[i]=rand()%2000+1;
bucket_sort( a, pom, n, key_min, key_max );
cout << "The sorted array is : \n";
/* for( index = 1; index <= n; index ++ ){
    cout << " " << a[index] << " :";};
    cout<<endl;*/

s=0;
for (i=1; i<= n;i++) { s=s+a[i];used[i]=0; };
piv=s/kmax;
for (k=1; k<=kmax-1;k++)
{
    delta=a[k]-piv;
    server[k]=a[k];
    if (a[k] >=piv) goto e10;
    dpmin=maxint;
    domin=-maxint;
    if (delta>=0) { dpmin=delta;
        jpmin=k;}
    else { domin=delta;jomin=k;used[k]=-k; };
    temp=a[k];
    j=kmax+1;
    while (j<=n)
    {
        if (used[j]==0 )
            {temp=temp+a[j];delta=temp-piv;
            if (delta>=0)
                //then
                { if (delta==0){ server[k]=piv;used[j]=k;goto e10; };

```

```

        if (delta<dpmin){ dpmin=delta;
            jpmin=j; };
            temp=temp-a[j];
        }
    else
        { if (abs(delta)<abs(domin)) { domin=delta;jomin=j ;};
            used[j]=-k;
        }
        //{end of if delta}
    };//{end of the first if};
    j=j+1;
    };//{end of the second loop - on j}
    if (abs(dpmin)<=abs(domin))
    { server[k]=piv+dpmin;
        used[jpmin]=k;
        for (jj=jomin; jj>= jpmin+1;jj--)
            if (used[jj]=-k)used[jj]=0;
            else server[k]=piv+domin;}
e10: ;
    };//end of the first loop - on k
    server[kmax]=a[kmax];
    for (j=kmax+1; j<=n;j++) if( used[j]==0)
    server[kmax]=server[kmax]+a[j];
    //{writeln; writeln('s=',s,' pivot=',piv);}
    snova=0;
    mmin=server[1];mmax=mmin;
    for (k=1;k<=kmax;k++ )
    {
        //{write(' k=',k, ' ',server[k]);}
        snova=snova+server[k];
        if (server[k]>mmax) mmax=server[k]; else if (server[k]<mmin)
            mmin=server[k];
        //{writeln;writeln('snova=',snova);}if snova<>s then
        writeln('error');
        //{writeln('misbalans=',mmax-mmin)}
    };
    // finish(t);
    // report('t=',t);
    cout<<"s="<<s<<" pivot="<<piv<<endl;
    cout<<"snova="<<snova<<endl;
    cout<<"misbalans="<<(mmax-mmin)<<endl;
}

```

Program 1- center for finding single center of graph

Short description

There are two major steps in this algorithm:

1. For each column j of the input matrix $W[i][j]$, find the maximum term in that column.
2. Among the maximum terms taken from each column, find the minimal one. That will be the 1-center of the given graph.

Code implementation in C++

```
// 1-center.cpp
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
    int **C;           // the matrix
    int array_size = 0; // matrix size
    // Data input
    cout << "Matrix size: ";
    cin >> array_size;
    C = new int* [array_size];
    for (int i = 0; i < array_size; ++i)
        C[i] = new int [array_size];

    cout << "Enter matrix elements:\n";
    for (int i_row = 0; i_row < array_size; ++i_row)
        for (int i_column = 0; i_column < array_size; i_column++)
            cin >> C[i_row][i_column];

    // Calculations
    // 1-center
    int center_row = 0;
    int center_value = 0;
    for (int i_column = 0; i_column < array_size; ++i_column)
    {
        int column_max_value = 0;
        int i_row_max = 0;
        for (int i_row = 0; i_row < array_size; ++i_row)
        {
            if (C[i_row][i_column] > column_max_value)
            {
                column_max_value = C[i_row][i_column];
                i_row_max = i_row;
            }
        }
        if (column_max_value < center_value || i_column == 0)
        {
            center_value = column_max_value;
            center_row = i_row_max;
        }
    }
}
```

```
        // Output
        cout << '\n';
        cout << "center_value = " << center_value << '\n';
        cout << "center_row = " << center_row + 1 << '\n';

        //Cleanup
        for (int i = 0; i < array_size; ++i)
            delete C[i];
        delete C;

        return 0;
}
```

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED "AS IS." EMC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.