The Barnes-Hut Algorithm in MapReduce

Ross Adelman
radelman@gmail.com

1. INTRODUCTION

For my end-of-semester project, I implemented an N-body solver in MapReduce using Hadoop. The N-body problem is a classical one in computational physics. Suppose there are N bodies on the 2D plane, each with a mass, m_n, and position, $\mathbf{x}_n$. The force exerted on the nth body, $\mathbf{f}_n$, by all of the other bodies is given by

$$\mathbf{f}_n = \sum_{k \neq n} \frac{m_n m_k \mathbf{r}_{nk}}{r_{nk}^2} \tag{1}$$

where $\mathbf{r}_{nk} = \mathbf{x}_k - \mathbf{x}_n$ and $r_{nk} = |\mathbf{r}_{kn}|$. Since this expression is evaluated for each body, and the expression contains a sum over every other body, a direct computation of the forces would require $\mathrm{O}\left(N^2\right)$. For large problems, such as those containing more than a few thousand bodies, this can be prohibitive. Therefore, a faster algorithm is needed.

The Barnes-Hut algorithm is one such algorithm [1]. The fast multipole method, introduced in [2], is another. The fast multipole method is quite complex, so I chose to focus on the Barnes-Hut algorithm. The multilevel Barnes-Hut algorithm is an $\mathrm{O}\left(N \log\left(N\right)\right)$ algorithm, but to keep things simple, I implemented a single-level version, which runs in $\mathrm{O}\left(N^{3/2}\right)$.

MapReduce has been used before to analyze N-body-like data. For example, in [4], a friends of friends algorithm was distributed across a MapReduce-like framework. Also, in [5], Pig was used to analyze large amounts of astronomical data. In both of these, the datasets were very large, in the hundreds of GBs and low TBs. These examples give hope that MapReduce can be used effectivly on a N-body solver.

Previous work in distributing N-body solvers is quite extensive. A recent example is [3], where a billion-body problem was solvable in seconds across a large hetergeneous CPU/GPU cluster. In these, the programmer must handle all aspects of the distribution, including synchronization, message passing, load balancing, and so on. In doing so, however, they are able to optimize extensively, achieving extremely fast solvers.

By implemeting a N-body solver in MapReduce instead of using another distribution framework, like MPI, quite a bit of speed is sacrificed, but for the benefit of easier development.

2. BARNES-HUT ALGORITHM

The single-level Barnes-Hut algorithm calculates an approximate solution to Eq. (1) in $\mathrm{O}\left(N^{3/2}\right)$ by separating the sum into two components, one that calculates the force due to nearby bodies, and one that calculates the force due to faraway bodies. In other words,

$$\mathbf{f}_n = \sum_{k \neq n, \mathbf{x}_k \in \mathrm{N}_n} \frac{G m_n m_k \mathbf{r}_{nk}}{r_{nk}^3} + \sum_{k \neq n, \mathbf{x}_k \notin \mathrm{N}_n} \frac{G m_n m_k \mathbf{r}_{nk}}{r_{nk}^3} \tag{2}$$

where N_n is the neighborhood of the nth body (a body in N_n is considered nearby the nth body). The second sum can be approximated in the following manner. Suppose there is a group of bodies far away from the body. The effect these bodies can be approximated by a new body, called a virtual body, whose mass is equal to the total mass of the group and position is equal to the center of gravity of the group.

The process of creating the virtual bodies is illustrated in Figure 1. To begin, the 2D plane is divided up into $K \times K$ boxes. For each box, the total mass and the center of gravity of the bodies in that box are calculated. A virtual body is created with that total mass and is placed at that center of gravity.

The process of calculating the force on a body is illustrated in Figure 2. The plot on the top shows the process when using Eq. (1). The force on the magenta-colored body is being calculated. The green-colored bodies are those that need to be considered when calculating the force. In this case, all of them. The plot on the bottom shows the process during evaluation when using the Barnes-Hut algorithm. Like before, the force on the magneta-colored body is being calculated. The box to which the body belongs and the neighboring boxes are chosen to be N_n, so the bodies in these boxes are used directly to calculate the force on the box. The bodies in the other boxes are not considered separately. Instead, the virtual bodies are used. Like before, the bodies used to calculate the force on the magneta-colored body are colored

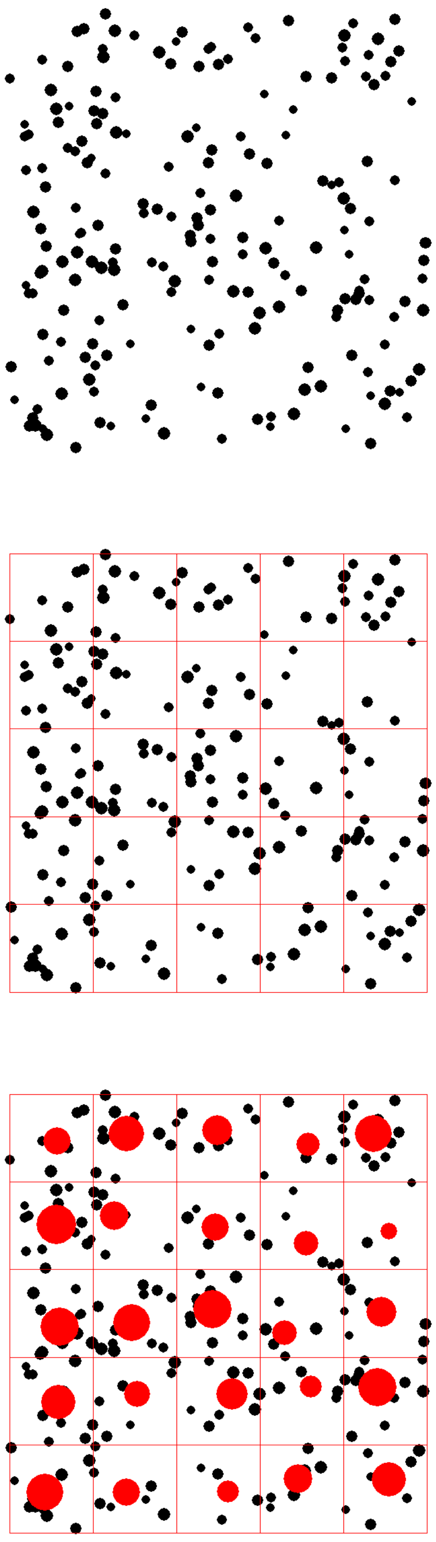


Figure 1: On the top: a sample distribution of bodies. In the middle: the domain is divided up into $K \times K$ boxes. On the bottom: a virtual body is calculated for each box.

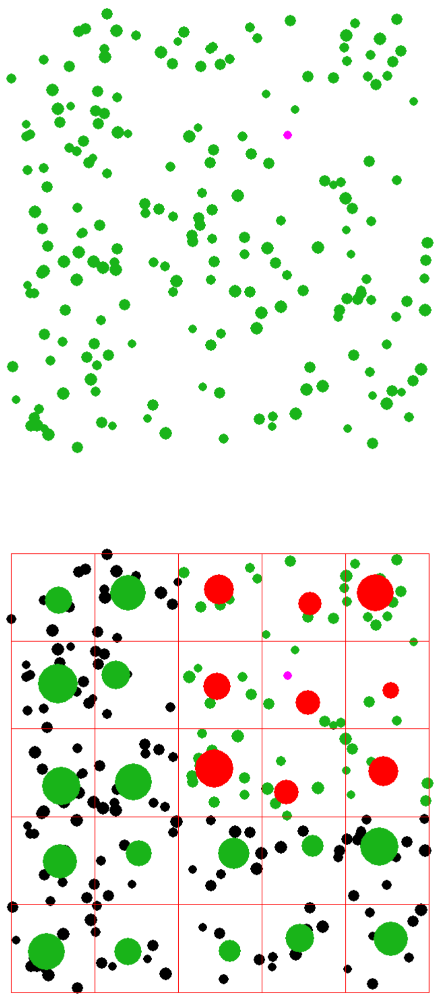


Figure 2: The green-colored bodies are used to calculate the force on the magenta-colored body. On the top: for direct evaluation, all of the other bodies must be considered. On the bottom: using the Barnes-Hut algorithm, only the bodies nearby the magenta-colored body must be considered directly; the virtual bodies can be used to approximate the effect of the far-away bodies.

in green.

3. COMPUTATIONAL COMPLEXITY

Let us count the number of body-body operations to calculate the computational complexity of the single-level Barnes-Hut algorithm. To begin, assume the bodies are uniformly distributed, so that there are roughly the same number of bodies in every box. For the nth body, the bodies in N_n and the virtual bodies outside of N_n are used to calculate the force. The number of bodies in N_n other than the nth body is $9N/K^2 - 1$. The number of virtual bodies outside of N_n is $K^2 - 1$. This is repeated N times, once for each body, so that the total number of body-body operations is equal to

$$T = N\left(\frac{9N}{K^2} + K^2 - 10\right) \tag{3}$$

To obtain an expression only in terms of N, we need to calculate an optimal K. To do this, note that T has the following form: for small K, T is very large. As K increases, T termporarily decreases to a minimum, but then increases again. Therefore, we can calculate the optimal K by finding the K that minimizes T. Differentiate T with respect to K, set the resulting expression equal to zero, and solve for K.

$$K = (9N)^{1/4} \tag{4}$$

Plugging this back into Eq. (3) yields

$$T = 2\sqrt{9}N^{3/2} - 9N \tag{5}$$

$$T = \mathrm{O}\left(N^{3/2}\right) \tag{6}$$

4. MAPREDUCE VERSION

The body information is stored on HDFS as comma-separated values in plain ASCII. Each line contains information for one body, and is formatted as

```
bodyID bodyM,bodyX,bodyY
```

Here are a few lines from some sample data.

```
1 17.9712380472,-0.888293927879,-0.845638591148
2 11.1535717816,-0.589199280636,-0.605215012814
3 17.6645610833,0.640402118083,-0.560898604327
4 18.8974057315,-0.489396557808,0.454717832501
5 19.6026673097,0.600943016412,0.156131080437
```

4.1 Mapper Phase

The mapper reads in the body information from HDFS line by line, and for each body, begins by parsing the information into a key-value pair.

```
<bodyID, (bodyM, bodyX, bodyY)>
```

To which box the body belongs is calculated as a pair of indices.

```
(bodyBoxI, bodyBoxJ)
```

A copy of the body information is sent to each reducer as a special key-value pair. The key is a triple of integers. The left-most integer indicates to which reducer the key-value pair should be forwarded, and the two right-most integers are the indices of the box to which the body belongs. The value is the body key-value pair from above. The left-most integer in the keys are made nonpositive for these special key-value pairs so that they arrive at the reducers before the regular key-value pairs. To improve performance, in my implementation, I use in-mapper combing to reduce how many of these I have to send.

```
<(0, bodyBoxI, bodyBoxJ),
         (bodyID, (bodyM, bodyX, bodyY))>
<(-1, bodyBoxI, bodyBoxJ),
         (bodyID, (bodyM, bodyX, bodyY))>
...
<(-numReducers, (bodyBoxI, bodyBoxJ)),
         (bodyID, (bodyM, bodyX, bodyY))>
```

Nine more copies of the body information are sent as regular key-value pairs, one to the box in which the body belongs, and eight to the neighboring boxes. The left-most integer of the key is equal to one so that they arrive after the special key-value pairs from above. The `bodyID` is negated for the neighboring boxes, indicating that the body does not belong to that box.

```
<(1, bodyBoxI - 1, bodyBoxJ - 1),
         (-bodyID, (bodyM, bodyX, bodyY))>
<(1, bodyBoxI - 1, bodyBoxJ),
         (-bodyID, (bodyM, bodyX, bodyY))>
<(1, bodyBoxI - 1, bodyBoxJ + 1),
         (-bodyID, (bodyM, bodyX, bodyY))>
<(1, bodyBoxI, bodyBoxJ - 1),
         (-bodyID, (bodyM, bodyX, bodyY))>
<(1, bodyBoxI, bodyBoxJ),
         (bodyID, (bodyM, bodyX, bodyY))>
<(1, bodyBoxI, bodyBoxJ + 1),
         (-bodyID, (bodyM, bodyX, bodyY))>
<(1, bodyBoxI + 1, bodyBoxJ - 1),
         (-bodyID, (bodyM, bodyX, bodyY))>
<(1, bodyBoxI + 1, bodyBoxJ),
         (-bodyID, (bodyM, bodyX, bodyY))>
<(1, bodyBoxI + 1, bodyBoxJ + 1),
         (-bodyID, (bodyM, bodyX, bodyY))>
```

4.2 Partitioner

When the key-value pair looks like

```
<(-reducerID, bodyBoxI, bodyBoxJ),
         (bodyID, (bodyM, bodyX, bodyY))>
```

the partitioner forwards the key-value pair to `reducerID`. Otherwise, the default hash partitioner is used.

4.3 Reducer Phase

The special key-value pairs arrive at each reducer before the regular ones. As they arrive, the reducer uses them to construct the virtual bodies. To begin, the virtual bodies have no mass and are located at the origin. The special key-value pairs arrive as

```
<(-reducerID, bodyBoxI, bodyBoxJ),
          (bodyID, (bodyM, bodyX, bodyY))>
```

For each one, the body's mass and position are added to the corresponding virtual body's mass and position.

```
boxM(bodyBoxI, bodyBoxJ) += bodyM
boxX(bodyBoxI, bodyBoxJ) += bodyM * bodyX
boxY(bodyBoxI, bodyBoxJ) += bodyM * bodyY
```

After all of the special key-value pairs have been processed, the virtual bodies' positions are divided by their masses.

```
boxX(bodyBoxI, bodyBoxJ) /=
          boxM(bodyBoxI, bodyBoxJ)
boxY(bodyBoxI, bodyBoxJ) /=
          boxM(bodyBoxI, bodyBoxJ)
```

Because I use in-mapper combining, many of these special key-value pairs are not distinct bodies, but rather partially computed virtual bodies. However, the above procedure does not change.

The keys of the regular key-value pairs arrive as

```
(1, bodyBoxI, bodyBoxJ)
```

and the values arrive as

```
(bodyID_1, (bodyM_1, bodyX_1, bodyY_1))
(bodyID_2, (bodyM_2, bodyX_2, bodyY_2))
(bodyID_3, (bodyM_3, bodyX_3, bodyY_3))
...
```

Remember, some of these `bodyID`s are negative, indicating that they belong to a neighboring box. The bodies are accumulated in an array. The force is calculated on each body in the array with a positive `bodyID`. First, the force due to the other bodies in the body's own box and the body's neighboring boxes is calculated. Second, the force due to the virtual bodies outside the body's neighborhood is calculated. These two forces are added, and a key-value pair is constructed and outputted. The key is the `bodyID`, and the value is a formatted string that contains the force on the body.

```
<bodyID, "bodyFX,bodyFY">
```

5. EXPERIMENTAL RESULTS

The code was developed on the Clouder VM, tested on the Hoth, a small, nine-node cluster made of old computers especially for the class, and deployed and experimented with on a cluster on Amazon EC2.

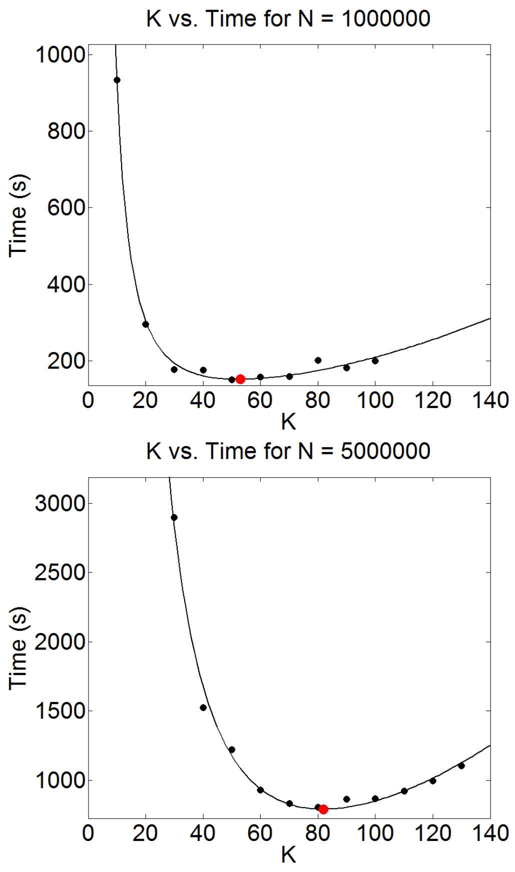
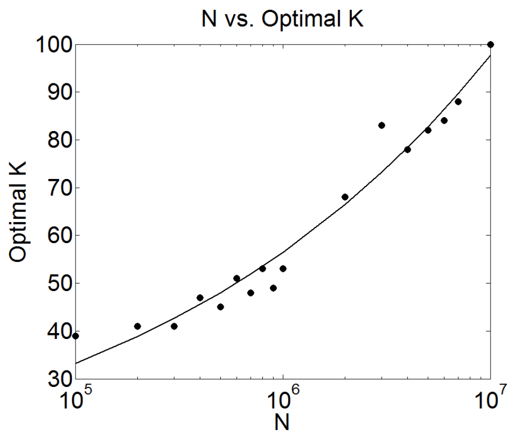


Figure 3: For each N, the optimal K was calculated experimentaly by running the code for several different values of K. The optimal K is where the running time is minimized.



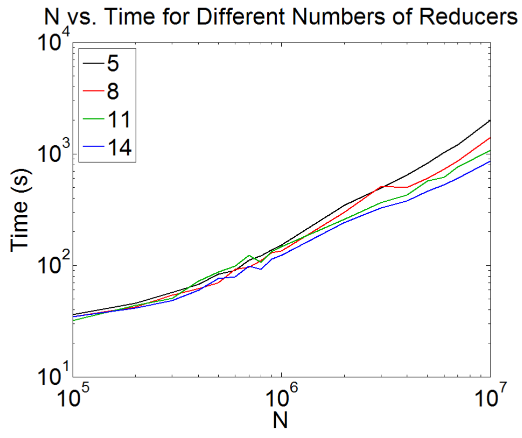


Figure 4: On the top: the optimal K as a function of N. On the bottom: the running time versus N for different numbers of reducers with the optimal K used.

For each N, the optimal K was calculated experimentally. The code was run for several different values of K, the running times were recorded, and these data were used to estimate the values of c_1, c_2, and c_3 in the following equation.

$$T = c_1 K^{-2} + c_2 K^2 + c_3 \tag{7}$$

Once c_1, c_2, and c_3 were known, the optimal K was calculated using

$$K = \left(\frac{c_1}{c_2}\right)^{1/4} \tag{8}$$

Figure 3 shows the above procedure for $N = 1000000$ and $N = 5000000$. The black circles are the running times recorded from running the code for several different values of K, the black curve is the equation in Eq. (7) after c_1, c_2, and c_3 were estimated, and the red circle is the minimum of that curve.

The above procedure was repeated for several different values of N from $N = 100000$ to $N = 10000000$. For each N, the optimal K was calculated experimentally. These data were used to estimate c_1 and c_2 in the following equation.

$$K = c_1 N^{1/4} + c_2 \tag{9}$$

The top plot in Figure 4 shows these data, as well as the curve in Eq. (9).

The code was run again for the same values of N using the optimal values of K. The runs were repeated using 5, 8, 11, and 14 reducers. The bottom plot in Figure 4 shows the running times for these runs.

The results were very good. The plots in Figures 3 and 4 match the theory well. For example, in Figure 3, the running time is high for low values of K, decreases quickly to a minimum, and increases slowly for higher values of K. As expected, the optimal K increases as N increases. In effect, the code attempts to balance the number of nearby bodies and the number of virtual bodies when calculating the force on a body. As N increases, K must also increase to maintain this balance. Based on the bottom plot in Figure 4, the running time appears to be slightly worse than $\mathrm{O}(N)$, but not as bad as $\mathrm{O}\left(N^{3/2}\right)$. This is likely due to the overhead from starting and stopping the MapReduce job in Hadoop, as well as the cost of send data over the network. As N increases, the running time will likely converge to $\mathrm{O}\left(N^{3/2}\right)$. Increasing the numbers of reducers appears to help, especially for larger values of N. For example, when $N = 10000000$, doubling the number of reducers seems to halve the running time.

6. CONCLUSION

A single-level version of the Barnes-Hut algorithm was implemented in MapReduce and executed on Hadoop cluster. Even though a simplified version of the algorithm was used, the results were very good, conforming to the theory well. The code was tested on data sets with up to 10000000 bodies. Even at that size, the code ran smoothly, even beating the predicted $\mathrm{O}\left(N^{3/2}\right)$ behavior. The implementation could be extended in a number of ways in the future. These include: encoding the body information in a more compact way (using binary instead of ASCII); using more compact intermediate key-value pairs (the current ones are fairly verbose); implementing the multilevel Barnes-Hut algorithm; and extending the framework to the fast multipole method, a closely related algorithm.

7. REFERENCES

[1] J. Barnes and P. Hut. A hierarchical $\mathrm{O}(n \log(n))$ force-calculation algorithm. *Nature*, 324:446 – 449, 1986.

[2] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[3] Q. Hu, N. A. Gumerov, and R. Duraiswami. Scalable fast multipole methods on distributed heterogeneous architectures. In *SC'11 Proceedings of the 2011 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[4] Y. Kwon, D. Nunley, J. P. Gardner, M. Balazinska, B. Howe, and S. Loebman. Scalable clustering algorithm for n-body simulations in a shared-nothing cluster. In *22nd International Conference on Scientific and Statistical Database Management (SSDBM), Heidelberg, Germany*, July 2010.

[5] S. Loebman, D. Nunley, Y. Kwon, B. Howe, M. Balazinska, , and J. P. Gardner. Analyzing massive

astrophysical datasets: Can pig/hadoop or a relational dbms help? In *In the Workshop on Interfaces and Abstractions for Scientific Data (IASDS) 2009, New Orleans, LA*, August 2009.