# Cloud Service Solving N-Body Problem Based on Windows Azure Platform

Dariusz Rafał Augustyn and Łukasz Warchał

Institute of Informatics
Silesian Technical University
Akademicka 16, 44–100 Gliwice, Poland
{draugustyn,lukasz.warchal}@polsl.pl

**Abstract.** This paper shows how to use cloud computing to solve N-body problem. It presents an idea and implementation of cloud service based on Windows Azure Platform. Clients can access cloud service via Internet over HTTP protocol. They create computation tasks supplying simulation parameters such as number of steps, time step and XML file with body definitions (initial position, mass and velocity). Presented solution uses Barnes-Hut Algorithm (based on adaptive oct tree) to reduce computation complexity form $N \times N$ to $N \log N$. All body interactions are computed in parallel, on worker nodes in cloud.

**Keywords:** distributed processing, cloud service, Microsoft Windows Azure Platform, continuous dynamical systems simulation, N-body problem.

## 1 Introduction

N-body simulation methods are fundamental in domains of modeling of complex systems e.g. astrophysics (gravity interactions), molecular dynamics (electrostatics – Biot-Savart and van der Waals interaction), computer graphics (radiocity metods - computer image generation). Modern N-body simulation applications allow to observe an evolution of complex $N$ bodies systems for huge values of $N$.

The most simple and naive method is based on an assumption that bodies interact with each other. For this method so-called all-paired one, a complexity of computational work scales asymptotically as $N^2$.

There are many methods which decrease this complexity. Most of them use one of two approaches: Barnes-Hut algorithm [1] or FMM (Fast Multipole Method) [2]. Those methods are efficient but they give approximate results.

In this paper besides all-paired method the Barens-Hunt algorithm is considered. In the BH method the 2-dimensional space or 3-dimensional one is divided using an adaptive quad tree or oct one (both called BH-tree). Calculation of interaction (i.e. a force vector) for a selected body is preformed using only BH-tree's nodes (no calculating components of the force for all $N - 1$ bodies). Because of the height BH-tree (significant less than $N$) this reduces the problem complexity so it can be computed in time as $\mathrm{O}(N \log N)$.

Because a task of calculation for a selected single body is independent of other tasks, all those tasks can be executed separately. This gives a possibility of efficient parallel implementations of N-body simulation methods using e.g. distributed calculation environments.

There was many approaches to solve problem of parallel execution of N-body simulation (e.g. [1,2,3,4,5,6,7]) but this paper shows the application of cloud computing by using the newest Microsoft distributing calculation environment – the Windows Azure Platform [8].

Nowadays cloud computing becomes more and more popular. Growing number of developers deploy their applications in Internet-accessible data centers and never again worry about infrastructure, availability or data security. Windows Azure Platform gives flexible environment for creating service based applications accessed by large number of users via Internet. This solution can be easily scale to fit particular requirements. It offers Data Storage capabilities that allow to handle huge binary data files and tables containing billions of entities (with simple REST [12] interfaces). Communication between different application components is realized by queues (also accessed RESTfully). Windows Azure allows to focus on solving main problem rather than building complex infrastructure (software and hardware) for application.

## 2 Applied N-Body Methods – The Theoretical Background

N-bodies problem will be explained by presenting some example of evolution an astrophysical star system in the 3-dimensional space – a spiral galaxy.

The system can be described as a set of bodies. $m$ – mass, $x$ coordinate, $y$ one, $z$ one, $v_x$ – $x$-component of velocity vector, $v_y$ – $y$-component of velocity, $v_z$ – $z$-component of velocity are 7 properties of each body form the given system.

The movement of $i$ body in $t_{k+1}$ (the next moment of time) using values in $t_k$ (the current moment) can be approximately described by following obvious forms (based on the second Newton's law of motion):

$$t_{i,k+1} = t_{i,k} + dt, \tag{1}$$

$$\begin{bmatrix} x_{i,k+1} \\ y_{i,k+1} \\ z_{i,k+1} \end{bmatrix} = \begin{bmatrix} x_{i,k} \\ y_{i,k} \\ z_{i,k} \end{bmatrix} + \begin{bmatrix} v_{x,i,k} \\ v_{y,i,k} \\ v_{z,i,k} \end{bmatrix} dt, \tag{2}$$

$$\begin{bmatrix} v_{x,i,k+1} \\ v_{y,i,k+1} \\ v_{z,i,k+1} \end{bmatrix} = \begin{bmatrix} v_{x,i,k} \\ v_{y,i,k} \\ v_{z,i,k} \end{bmatrix} + \frac{1}{m_i} \begin{bmatrix} F_{x,i,k} \\ F_{y,i,k} \\ F_{z,i,k} \end{bmatrix} dt \tag{3}$$

where $[F_{x,i}\ F_{y,i}\ F_{z,i}]^T = \boldsymbol{F_i}$ is the resultant force vector.

For selected $i$ body the resultant force $\boldsymbol{F_i}$ is calculated using gravity interactions with all $N - 1$ remaining bodies.

The gravity interaction between $i$ body and $j$ one can by expressed as follows (the Newton's law of gravitation):

$$\boldsymbol{F_{ij}} = \begin{bmatrix} F_{x,ij} \\ F_{y,ij} \\ F_{z,ij} \end{bmatrix} = G\frac{m_i m_j}{r_{ij}^3} \begin{bmatrix} x_j - x_i \\ y_j - y_i \\ z_j - z_i \end{bmatrix} \quad , \tag{4}$$

$$r_{ij}^3 = (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 \tag{5}$$

where $G$ is the gravitational constant and $r_{ij}$ is the distance between $i$ body and $j$ one.

So the resultant force vector for single $i$ body can be obtained as follows:

$$\boldsymbol{F_i} = \sum_{i \neq j} \boldsymbol{F_{ij}} \quad . \tag{6}$$

Based on rules presented above (Equations (1)–(6)) the application for simulating gravity interactions was designed and developed. For example it was used for obtaining the simulation result of an evolution of a small spiral galaxy containing 100 bodies. A startup bodies configuration (position and velocity vectors of bodies) is shown on Fig. 1. The final bodies positions after 20 000 steps of the simulation (with $dt = 0.001$) is shown on Figs. 2 and 3. The system is unstable – some bodies "escaped from the galaxy" (Fig. 2). But most of bodies remained in the space region determined at startup (Fig. 3). Startup positions, final ones and trajectories of bodies are presented on Fig. 3.

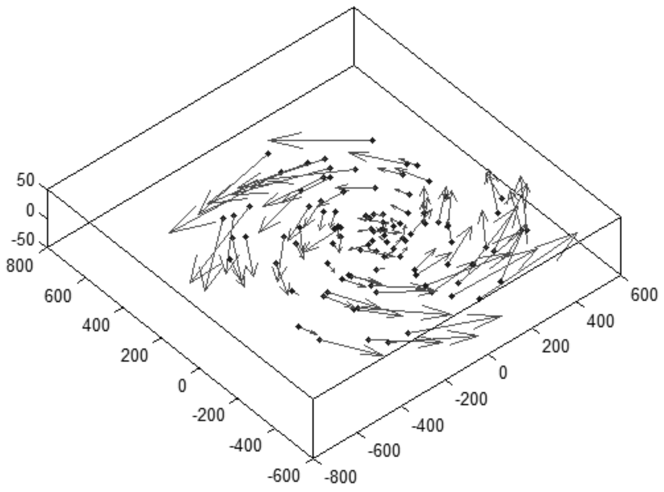


**Fig. 1.** View of startup position and velocity distribution for the sample spiral galaxy

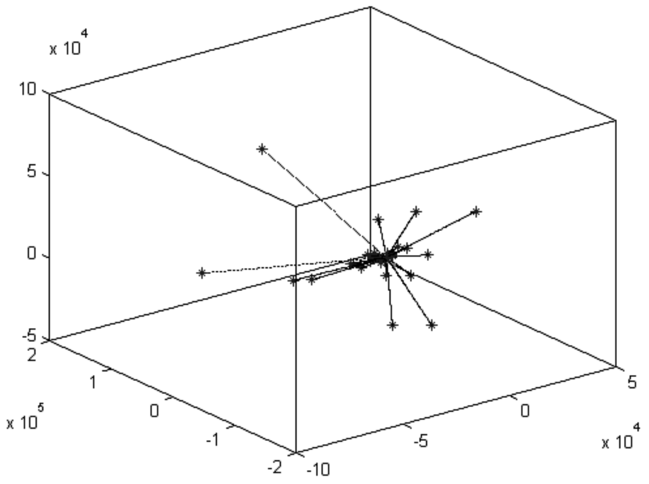


**Fig. 2.** Simulation results: escape of some bodies form the sample spiralgalaxy; final positions denotes by asterisks

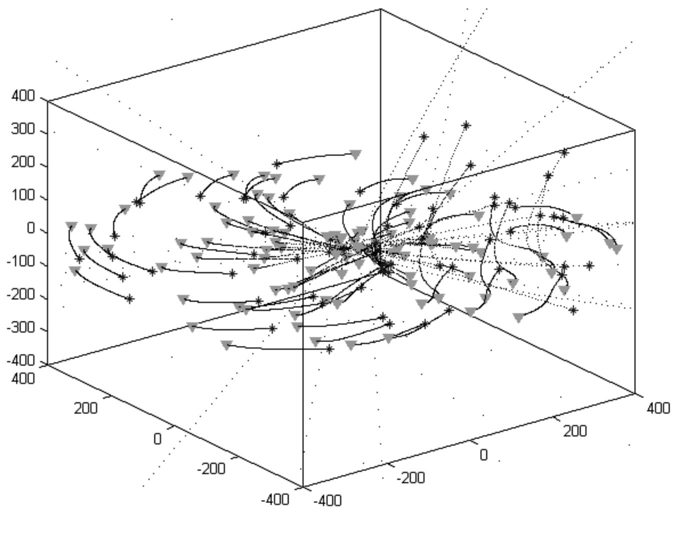


**Fig. 3.** Simulation results: bodies startup positions (triangles), final positions (asterisks) and trajectories in space located near the center of the sample galaxy

Because of efficiency reason the known approach [1] based on hierarchical tree data structures was considered too. For simplicity of presentation, the 2-dimensional case with a quad tree is shown at the first. The sample set of 5 bodies A,. . . ,D is located in nested squares (Fig. 4b). The space decomposition is finished when only one body is placed in a square. The process of this space division is illustrated on Fig. 4a.

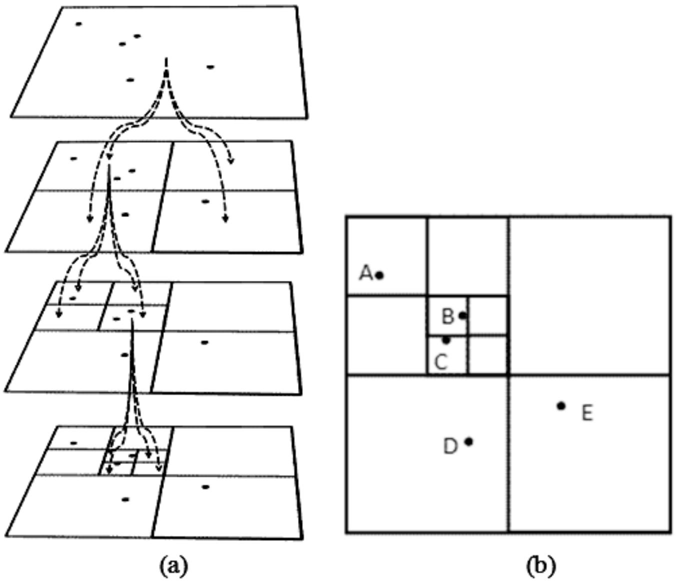


**Fig. 4.** Division of 2-dimensional space for sample set of bodies (division process – (a), result of division – (b))

The structure of sample adaptive quad tree is shown on Fig. 5a. Each tree node is identified by unique binary code. The method of coding nodes at every level of the tree is presented on Fig. 5b.

A similar method can be applied for a division of the 3-dimenasional space using an oct tree. Each node of the oct tree represents a cube. Each cube is decomposed on eight sub-cubes at the lower level of the tree. The method of identifying nodes in oct tree is shown on Fig. 6.

Advantages of the BH method results from applying some approximate method of calculation the resultant force. If a given body is distant enough from a node (square or cube) we can assume some simplification [1]. Let $d$ denotes the edge of a node (side of a square or edge of a cube). Let $r$ denotes the distance between the given body and center of mass (CM) of bodies subsystem nested in a node (see Fig. 7).

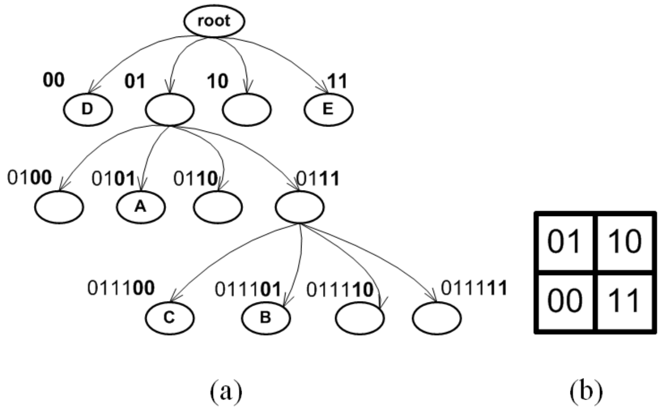


**Fig. 5.** Adaptive quad tree for sample bodies set with coded nodes – (a). Method of binary clockwise coding of quad tree's nodes – (b)

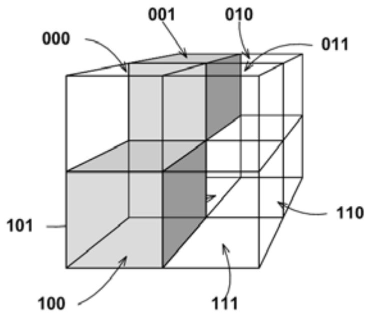


**Fig. 6.** Method of coding sub-cubes in the oct tree-based process of decomposition of 3-dimensional space

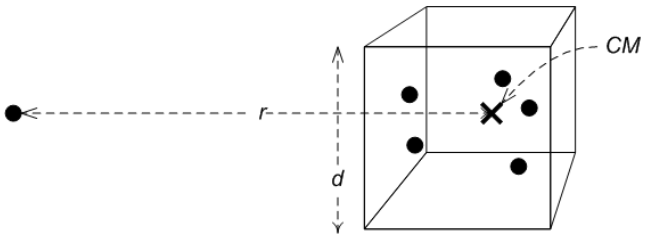


**Fig. 7.** Calculation of approximate resultant force between a single body and all bodies nested in a node

If formula presented below is satisfied (the long distant condition):

$$\frac{d}{r} < \theta \tag{7}$$

we can approximate the interaction between the given body and the subsystem nested in the node, by calculation of the resultant force as an interaction between only 2 bodies: the given body and a substitutionary body (with the mass equals to total mass of bodies nested in the node).

The $\theta$ is a parameter of the BH algorithm which determines an accuracy of the force approximation. It determines also the efficiency of the algorithm. Commonly values of $\theta$ belong to $[0.7, 1]$.

The method of calculation of the resultant force between the given $i$ body and $j$ node can be described by the following pseudocode:

```
function ResultantForce(i,j) {
    if (j is a leaf )
        return (force between i body and j one)
    elseif ( i body is distant enough from j node)
        return (approximate force between i body and j node)
    else {
        foreach ( l in sub-nodes(j)){
           if (l is not empty)
              aggregate ResultantForce(i,l)
        }
        return (aggregated resultant force)
    }
}
```

Calculation the interaction between a single $i$ body and all bodies of system is equivalent to invoking `ResultantForce (i, root of the tree)`.

## 3 Azure Platform Overview

Microsoft Azure Platform is an environment for running cloud applications and services. It has three main parts: *Compute* service, in which applications run, *Storage* service, which provides data storage and *Fabric* that manages and integrates all of them.

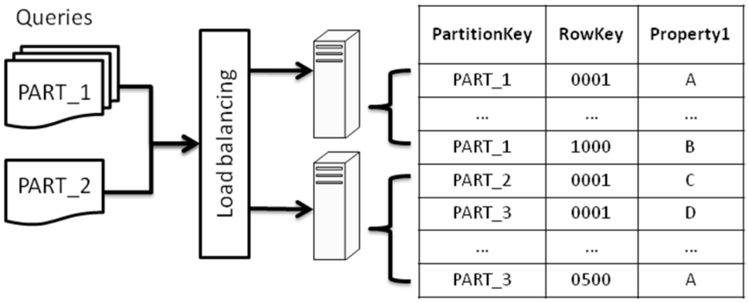
In Windows Azure each application can be made of two different instance types: Web role or/and Worker role. First one is responsible for interaction with end users. It can accept incoming HTTP or HTTPS requests. Typically, this is an ASP.NET web application or some kind of Web Service exposed by WCF [13]. Worker role instances are mainly considered as background processing components communicating with other workers or web role instances via queues.

Each Worker or Web instance runs inside its own Virtual Machine thus it is completely isolated. Number of instances of particular type is set at the stage of configuring application and can be change at runtime to avoid overloading. Each application needs to store its data in secure, reliable place, so Windows Azure provides mechanisms for storing and retrieving data in cloud-based environment. There are three kinds of Windows Azure storage: blobs, tables and queues, all accessed over standard HTTP protocol (using GET, PUT, DELETE methods).

## 3.1 Azure Table Storage

Azure Table Storage is a structured storage that supports scalable tables, that can contain large number of entities (rows) and terabytes of data [9]. It is efficiently scaling out by spreading to many servers as traffic grows.

Entity stored in a Table contains a set of user defined Properties (columns) and a PartitionKey and RowKey Properties that are used as primary key (both strings). PartitionKey is also used in scaling out. If there are many queries for entities from the same partition (the same PartitionKey), load balancing algorithm implemented in Windows Azure can activate dedicated node for serving only this partition, as shown on Fig. 8. Choosing proper way of creating partition key values is necessary to achieve good scalability.



| PartitionKey | RowKey | Property1 |
|---|---|---|
| PART_1 | 0001 | A |
| ... | ... | ... |
| PART_1 | 1000 | B |
| PART_2 | 0001 | C |
| PART_3 | 0001 | D |
| ... | ... | ... |
| PART_3 | 0500 | A |

**Fig. 8.** Scaling out with PartitionKey

All Tables in Windows Azure Table Storage have flexible schema, so two entities stored in one table can have different set of properties. This allows to avoid problems when entity definition is extended.

## 3.2 Azure Blob Storage

As mention above Azure Table Storage allows to keep terabytes of data in tables containing entities. But each entity can have maximum 255 different properties and only 1MB of combined size. Fortunately Windows Azure provides another mechanism for storing large binary data – Azure Blob Storage [11].

In Azure Blob each stored object can have up to 50 GB, it is replicated at least 3 times and strong consistency is provided. Blobs can be accessed from any place in any time using HTTP/REST protocol. The URI for particular object has following form:

```
http://<account>.blob.core.windows.net/<container>/<blobname>
```

where `<account>` is an account name in Windows Azure service, `<container>` is a name of container grouping a set of blobs (i.e. movies, pictures, etc.), and `<blobname>` is name of requested object.

### 3.3 Azure Queues

Azure Queues provides efficient and reliable message delivering mechanism [10]. It allows separated Worker and Web roles to communicate and exchange some data. Monitoring queue length can help deciding to increase number of processing instances.

All operations on queues can be done using HTTP REST protocol. An example of a GET request listing messages in a queue is presented below:

```
http://sampleAccount.queue.core.windows.net/nbodyqueue/messages?
timeout=30&numofmessages=32&peekonly=true
```

## 4 N-Body Cloud Service Architecture

Cloud service described in this paper is composed of Web and Worker role instances communicating over queues and storing data in Data Storage (Table and Blob) as shown in Fig. 9.

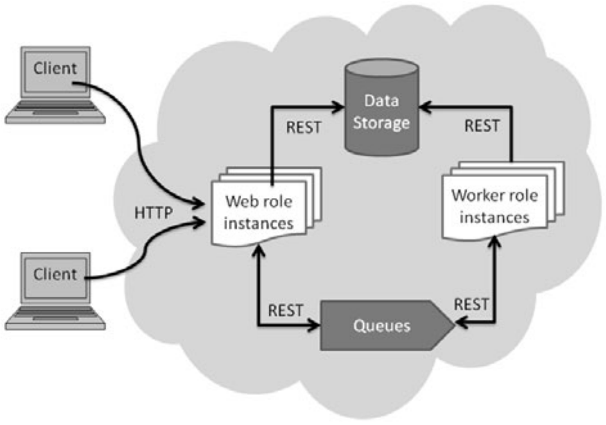


**Fig. 9.** Cloud Service Architecture Overview

Web role is a simple ASP.NET application. On main web page user defines simulation parameters such as: number of steps, time step, $\theta$ value and uploads XML file with bodies definitions (initial position, body mass and velocity), which is saved in Azure Blob Storage with an unique file name. After starting simulation, web application creates a *Task* entity (containing user supplied parameters) and stores it in *Tasks* table in Azure Table Storage. This entity is identified by unique string (Task Identifier) which is also sent back to client. After this, web application sends message to queue that worker instances can consume and start computation.

Worker role instances are responsible for some background processing and computation. Each instance reads a message from queue and depending of its type performs some task. The single message is a XML serialized object of class *NBodyWorkItem*, where value of *Type* property tells worker what to do. There are 4 main tasks that worker instances perform.

The presented cloud service works according to algorithm shown on an activity diagram from Fig. 10.

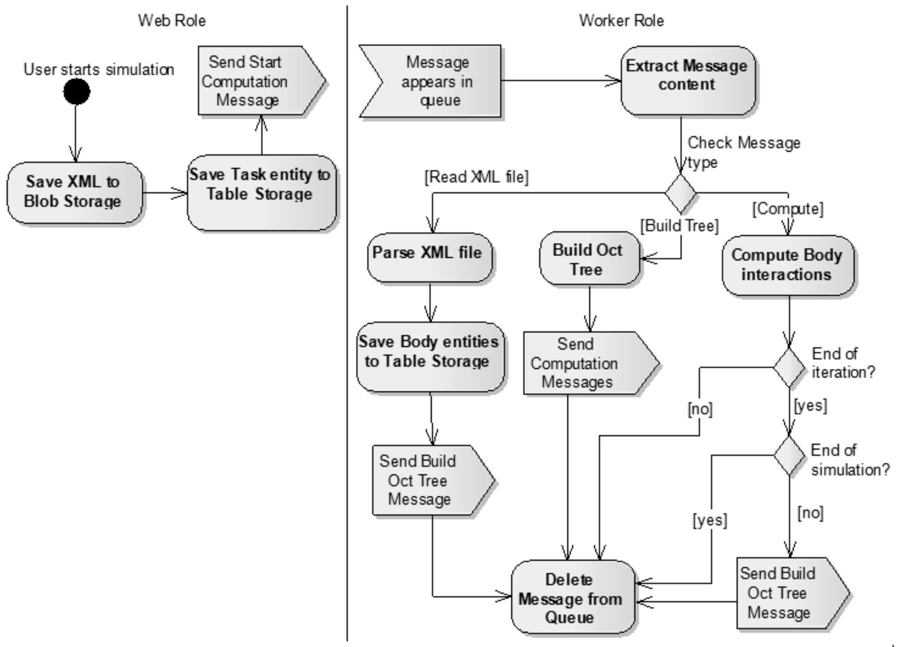


**Fig. 10.** Activity diagram presenting algorithm of N-body cloud service

When simulation starts, the first task is dedicated to parse uploaded by the user XML file. It was saved (by web application) in Azure Blob Storage. One of worker instances reads this file (its name is passed as a part of queue message), parses it and creates all *Body* entities that are persisted in Azure Table Storage in *Bodies* table. Task Identifier is used as a PartitionKey, so all *Body* entities from one simulation are in the same partition so they can be accessed in an efficient way. RowKey is a concatenation of iteration number (1 in this case) and some generated body identifier, so it is possible to distinguish between entities from different iterations.

Next one of worker instances is responsible for building oct tree. Each node in a tree is an *OctTreeNode* entity and it is stored in Azure Table Storage in

*OctTreeNodes* table. Oct tree is a hierarchical data structure but in this case it is persisted in flat table. To provide efficient way to access child nodes from parent one, special nodes identifying was implemented. This method of node coding is different from the one described in previous section because it is adapted to Azure's specificity. Each node has an unique identifier (RowKey property) which is a number converted to 20 characters length string padded with "0" to its left. Root has always RowKey equals to "0". Child nodes identifiers are generated according to this procedure written in C#:

```
for (int i = 1; i <= 8; i++) {
 // ...
 newId = String.Format("{0:D20}",Convert.ToInt64(parentId+i.ToString()));
 // ...
}
```

Such node coding makes that preparing LINQ query [14] against Azure Table Storage that finds all child nodes of given parent node is simple:

```
Int64 value = Convert.ToInt64(node.RowKey + "1");
string startOfLevel = String.Format("{0:D20}", value);
value += 7;
string endOfLevel = String.Format("{0:D20}", value);

var q = from n in DataContext.OctTreeNodes
        where n.PartitionKey == TaskId &&
              n.RowKey.CompareTo(startOfLevel) >= 0 &&
              n.RowKey.CompareTo(endOfLevel) <= 0
        select n;
```

When oct tree is ready, Worker sends $N$ messages (requests of computation tasks) to queue, one for each body. Other workers consume one of this messages and perform computation of body interaction using the oct tree (in parallel). It is obvious that increasing number of Workers (up to $N$) reduces the time needed to calculate particular iteration.

When new position and velocity of a body is known, new *Body* entity with this values is created and stored in *Bodies* table. When all bodies have been processed next iteration is started in similar order, unless it was the last one.

In the presented solution *Bodies* table contains for each body its "snapshots" from any iteration. This allows a user to download (as XML file) any step of simulation. The result XML file can be used in future as an input file for another simulation (or it can be presented to user as it was shown on Fig. 1).

## 5 Conclusions

This paper presents an idea and implementation of cloud service for N-body problem running on Windows Azure Platform. It covers various Windows Azure aspects and capabilities. Azure Storage is described in details. Presented service solves N-Body problem using all-pairs method or adaptive oct tree (Barnes-Hut Algorithm). Interaction for single body is now computed in parallel by Worker instances, however building oct tree is done by only one of them. Implementing parallel tree construction algorithm can obliviously help improving overall efficiency.

During the tests, we found that accessing Azure Storage is time expensive (it is foreseeable, because it is done with HTTP REST). Better efficiency can be achieved by reducing number of storage access, i.e. during body interaction calculation, each Worker can process more than one body at one time (one storage access returning many *Body* entities). For this reason we provided two implementations of building oct tree procedure. If number of bodies is not large, whole tree is first constructed in operating memory and then persisted in Table Storage in a single request. If number of bodies is very large, each successive oct tree nodes are immediately persisted to storage.

## References

1. Barnes, J., Hut, P.: A hierarchical $O(N \log N)$ force calculation algorithm. Nature 324 (1986)
2. Greengard, L., Rokhlin, V.: A Fast Algorithm for Particle Simulations. Journal of Computational Physics 73(325) (1987)
3. Warren, M.S., Salmon, J.K.: A parallel hashed Oct-Tree N-body algorithm. In: Proceedings of the 1993 ACM/IEEE conference on Supercomputing, pp. 12–21. ACM, New York (1993)
4. Dubinski, J.: A Parallel Tree Code, Santa Cruz (1996)
5. Cruza, F.A., Barba, L.A., Knepley, M.G.: Fast Multipole Method for particle interactions: an open source parallel library component. In: The 20th Parallel Computational Fluid Dynamics conference, Lyon (2008)
6. Izaguirre, J.A., Hampton, S.S., Matthey, T.: Parallel multigrid summation for the N-body problem. Journal of Parallel and Distributed Computing 65(8), 949–962 (2005)
7. Nyland, L., Harris, M.: Fast N-Body Simulation with CUDA. In: GPU Gems, vol. 3. Addison-Wesley Professional, Reading (2007)
8. Chappell, D.: Introducing Windows Azure. David Chappell & Associates (2009)
9. Haridas, J., Nilakantan, N., Calder, B.: Windows Azure Table (2008)
10. Calder, B.: Windows Azure Queue (2008)
11. Calder, B., Wang, T., Mainali, S., Wu, J.: Windows Azure Blob (2009)
12. Richardson, L., Ruby, S.: Restful Web Services. O'Reilly Media, Sebastopol (2007)
13. Peiris, C., Mulder, D., Cicoria, S., Bahree, A., Pathak, N.: Pro WCF: Practical Microsoft SOA Implementation. Springer, New York (2007)
14. Rattz, J.C.: Pro LINQ: Language Integrated Query in C# 2008 (Windows.Net). Apress (2008)