Cloud Based Decision Support System for Urban Numeric Data

Anthony Boulmier, Nabil Abdennadher, Gilles Desthieux, Claudiò Carneiro

IceBOUND final report

Thursday, June 9, 2016

REPUBLIQUE
ET CANTON
DE GENEVE

POST TENEBRAS LUX

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

SITG | LE TERRITOIRE GENEVOIS À LA CARTE

Contents

1 Introduction

Today cities of developed countries are increasingly digitized as 3D urban numerical models. However, the use of these models as a technological support for different applications related to the fields of environment, energy and urban planning, remain under-utilized. The assessment of solar potential (solar mapping) and the estimation of satellite visibility, underlined here as solar energy potential and Global Navigation Satellite System (GNSS) visibility respectively, are two types of applications based on the computation of 3D urban numerical models. The main goal is to use each of these applications for different stakeholders, such as:

1. Local collectivities, considering distinct needs and purposes;
2. Industrial and energy companies doing business at local collectivity scales;
3. Surveying companies that use GNSS for topographical measurements in dense urban areas.

The extraction of three-dimensional spatial indicators adjusted to both applications is a research challenge. It is done using different methods of aggregation of data. Indeed, the use of these particular spatial indicators allows a more refined analysis of the urban fabric and guarantees that output results are adjusted to end-user needs.

The goal of the "Cloud Based Design Support System for Urban Numeric Data" (iCeBOUND) project is to design and develop a Decision Support System (DSS) that leverages 3D digital urban data to facilitate environmental analyses in cities. This project aims at providing decision makers with relevant indicators for city planning and energy management;

Although both applications (solar energy potential and GNSS) are very different in terms of functionality and targeted market, the algorithm they use is the same: the Shading Algorithm as shown in Figure 1. This algorithm is memory and CPU time consuming.

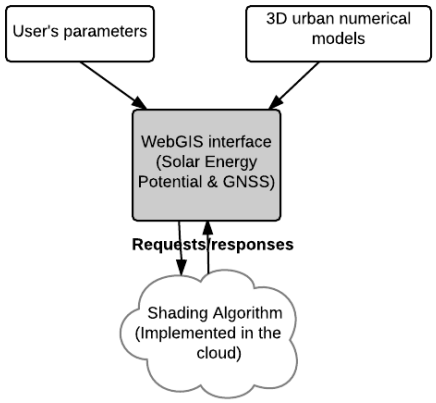

Figure 1 Two specific applications, one generic platform.

This report is organized as fellows: Section 2 describes the two use-cases targeted by the iCeBOUND project. It focuses on the end-user features that should be fulfilled by the two applications. Section 3 deals with the "Shading Algorithm ". It describes two versions of the algorithm. The goal is to compare them and select the most "efficient" and "cheapest" version. Section 4 (resp. Section 5) describes the dedicated algorithms used in the "solar energy radiation" (resp. GNSS visibility) application. Section 6 describes the parallelization of the Shading Algorithm and its deployment on Cloud infrastructures.

2 Use cases

This section is dedicated to the presentation of the two use cases developed during the iCeBOUND project. The first use case is an efficient solution for providing decision makers for city planning and energy management whereas the second one is a tool for quantifying the satellite signal quality in urban area for planning field measurements.

2.1 Solar potential

Recently increasing attention given to environmental issues in urban studies shows that city planners are convinced that cities play a leading role in controlling sustainability. In addition, industrial companies showed an increasing interest in satellite signal quality estimation in order to plan efficient field measurements.

The first use case, "Solar Energy Potential", consists of evaluating the potential of house and building roofs located in urban areas for producing solar energy (both thermal and photovoltaic). The creation of solar maps of cities allowing to accurately evaluate areas that can be used for the installation of renewable energy, such as solar panels on building roofs, is currently considered as highly relevant [1]. This provides an automated and dynamic solar radiation calculation in order to provide an accessible platform for many different stakeholders. The goal of the solar potential use case is to develop a decision support system that leverages 3D urban data to facilitate environmental analysis of cities or regions. In other words, this project aims at providing relevant ecological information for efficient solar panel installations in cities.

For this purpose, the solar energy potential is calculated by summing the estimated value of the solar radiation in kWh by hour, for every hour during a time span at a given place. To compute accurately the amount of solar energy that can be produced by a solar panel, two families of parameters have to be taken into account:

1. Natural parameters: weather, sky visibility and sun position;
2. 3D urban numeric data.

Weather is represented by a weather data file, which is a set of coefficients based on the previous years measurements of each month. This information allows to take roughly into account the typical weather of each month. The sky visibility (Sky View Factor - SVF) is calculated by a dedicated algorithm, which will be detailed later. Concretely speaking, solar radiation is computed using:

1. The location (in latitude);
2. The weather;
3. The surrounding relief and landscape: mountains, hills, etc;
4. 3D urban numeric data: buildings' structure
5. The distant and close hourly shading (at a given position);
6. The ratio of the visible sky (at a given position): the "sky view factor" (SVF).

The four first points are input data while the two last points are calculated thanks to the Shading Algorithm. This algorithm is observed to be irregular and time consuming.

The cities are digitized and divided into several sub-areas called tiles. Each tile is represented by a matrix that describes the targeted sub-area. Matrices are stored in a Geolocalized TIFF format (GeoTIFF):

- The Digital Urban Surface Model (DUSM) that describes the elevation of the buildings on each point of a given tile. This file has a resolution of 0.5 m. It represents the structure of the city such as the buildings and the houses for calculating close shadings. The values of this file are encoded in 32 bit floating point numbers.

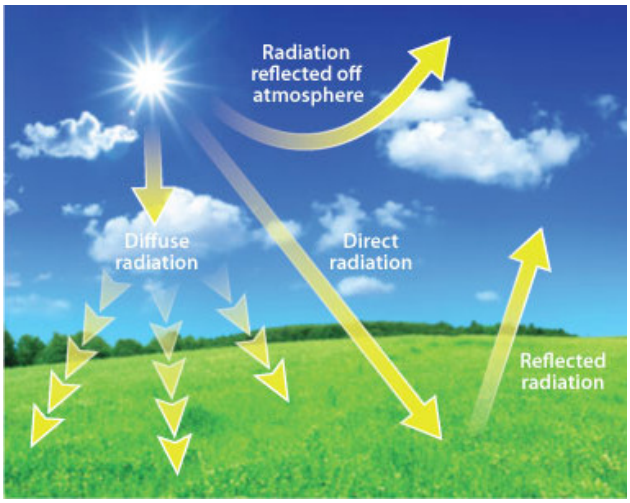

Figure 2 Three components of the solar radiation: direct, reflected and diffuse radiation.

- The Digital Terrain Model (DTM) which describes the elevation of the ground on each point of a given tile. This file has a resolution of 50 m. It represents the terrain relief such as the mountains for calculating distant shading. The values of this file are encoded in 32 bit floating point numbers.
- The slope matrix which describes the slope of each point (value between 0 and 90°) that belongs to a roof of a given tile. A point with a slope of 0° belongs to a flat roof whereas a point with a slope of 90° belongs a facade. This file is used to understand more accurately the structure of the tile surface. The values of this file are encoded in 8 bit integers.
- The orient matrix which describes the orientation of each point (value between 0 and 360°) that belongs to a roof of a given tile. A point with an orientation of 0° is north oriented whereas a point east oriented has an orientation of 90°. That is, the orientation is growing in clockwise direction. The values of this file are encoded in 8 bit integers.

To read and write the GeoTIFF data, "jai-ImageIO" [2] has been used. This library allows the reading and the writing of complex image formats such as GeoTIFF.

The solar radiation is the sum of three components: The direct solar radiation, the diffuse solar radiation and the reflected solar radiation. The direct radiation is directly proportional to the sun visibility. The reflected radiation depends on the ground and the nearby object reflection. The diffuse radiation is derived from the sky visibility. Figure 2 illustrates the three components of the solar radiation that are involved in the calculation of the solar energy potential.

The altitude and azimuth of the sun (i.e the sun position) is used to compute the direct solar radiation. The sky visibility (or Sky View Factor - SVF) is used to compute the diffuse radiation. SVF describes the amount of visible sky from a point. SVF is calculated by using a sky model. The sky model represents the sky as a cupola composed of different light sources (the more light sources are in the sky model, the more accurate will be the SVF). A light source is given by its azimuth and altitude component. For instance, a point located in an urban area has a SVF close to 0 whereas a point located in a rural area has a SVF close to 1. Figure 3 illustrates the above example.

The outcome of "Solar Energy Potential" is a solar map of the given tile which corresponds to the level of solar energy that can be produced by a solar panel on each point of the tile in kWh. The values of these maps are all encoded in a 64 bit floating point GeoTIFF format. Figure 4 shows the result of a yearly solar radiation study of the residential zone of Meyrin, Switzerland. Bright points are sunny areas that have a significant solar potential.

Figure 3 Illustration of the sky view factor in a rural and an urban environment.

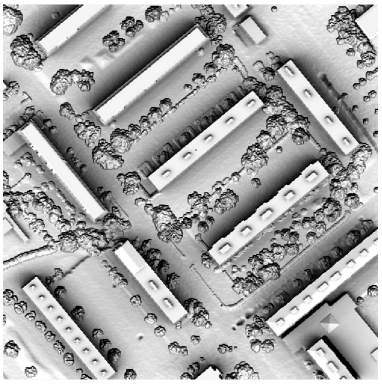

Figure 4 Yearly solar energy map of the residential zone of Meyrin, Switzerland.

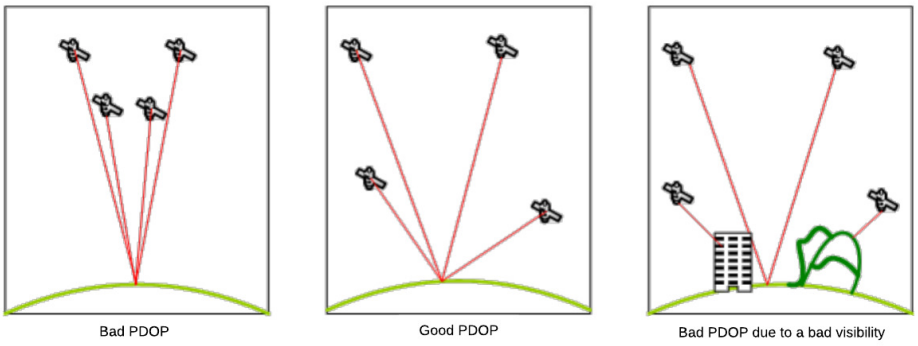

Figure 5 Bad and good PDOPs

In general, roofs are the most efficient place to put solar panels in order to generate solar energy. Nevertheless, when facades are south oriented, they also provide a good place to install a solar panel. Therefore, solar radiation maps are also carried out for facades by extending the "Solar Energy Potential" use-case.

The facades of a given tile are described using these descriptors:

- The "front id" matrix which contains the unique identifier of each facade of a given tile. These values are encoded in 32 bit integers. To clarify, each point P that belongs to a tile has an unique facade identifier, which is an integer between 1 and 2^{32} (i.e the unique facade id) if the point belongs to a facade and 0 otherwise.
- The "front orient" matrix which gives the orientation of each point that belongs to a facade of a given tile between 0 and 360°. A facade with an orientation of 0° is north oriented. The values of this file are encoded in 8 bit integers.
- The "front DTM" matrix is the DTM file described above with the resolution of the DUSM. The values of this file are encoded in 32 bit floating point numbers.

2.2 Global Navigation Satellite System (GNSS)

The second use case, Satellite Visibility, focuses on quantifying the number and position (overall distribution, also called PDOP : Position Dilution of Precision) of available satellites in the sky at a given time. The main goal of this decision support system is to optimize human resources and equipment needs for topographic measurements to be done on urban areas during a certain period of time. GPS, GLONASS and Galileo satellites constellations can be used to calculate satellite visibility and PDOP.

The GNSS application aims to provide accurate information of the quality of the PDOP during a given time span. Figure 5 shows different satellites typologies that result in a good or bad PDOP.

As each satellite movement in the sky behaves like the sun, the Shading Algorithm can be also applied for calculating visibility of each satellite from any line of sight from the surface of the earth. To reach this goal, cities are, like for the use case of solar potential presented in Section 2.1, digitized as DUSM. Hence, users can dynamically select an area of the DUSM in order to perform computation. Information needed to calculate satellite visibility and PDOP at a given place and time is given by an almanac, provided by GNSS Trimble Company. This almanac:

- provides orbital parameters (also called keplerial) for each satellite,
- is organized by satellites constellation (GPS, GLONASS, etc.),

- is daily updated and published as a text file in Trimble Company website.

The duration of calculations of satellite visibility and PDOP is made using a starting date, a number of forecasts and a spacing (e.g., every 10 minutes) between forecasts. The number of forecasts gives the number of shading and PDOP independent calculations to perform, which are based on three distinct processes:

- calculation of satellites positions: involves a specific transformation of the almanac data and the location (i.e latitude and longitude geodetic coordinates) of satellites to distinct azimuths and altitudes positioning,
- calculation of satellites visibility on each point of the DUSM at a given time: implies one shadow process for each satellite available in the sky in order to determine if a satellite is hidden or not by existing obstacles in the urban fabric. The result of each satellites visibility forecast for a given time is presented as a map, formatted as a GeoTIFF 32 bit floating point numbers,
- calculation of PDOP: corresponds to overall satellites visibility positioning quality on each point of the DUSM at a given time. The result of each PDOP forecast is presented as a map, formatted as a GeoTIFF 64 bit floating point numbers.

To summarize, updated almanac information and location of the area to be analyzed are used to calculate the position of satellites that are available in the sky. Using topocentric coordinates (azimuth and elevation) of available satellites, WGS84 coordinates of GNSS position receiver and DUSM it is possible to calculate which satellites are visible from any line of sight, according to natural morphology and artificial objects that exist in the urban fabric. Finally, using East North Up (ENU) coordinates of visible satellites and GNSS position receiver it is possible to compute PDOP.

3 Shading Algorithm

This section explains the Shading Algorithm used in the GNSS and solar potential applications. This algorithm calculates the shadow map of a given tile according to a given light source (sun in case of solar potential application) or signal source (satellite in case of GNSS application). This map contains Boolean values that represent the shading of each point (0 for shaded, 1 otherwise) for a given light/signal source. Figure 7 shows three calculated shadow maps for an area of Geneva at 8 am, 2 pm and 5 pm.

The purpose of a Shading Algorithm is to discover, for each point P that belongs to an urban model (i.e DTM or DUSM), which other point P' is shading it according to a given light/signal source. Figure 6 shows the algorithm explained above. When the sun is in position 1, H1 is higher than P' and P is sunny. On the other hand, when the sun is in position 2, the height H2 is less than that of the building (point P'), P is shaded.

Therefore, the shadow algorithm can be formalized by the following equation:

$$S_l(P_0) = ! \bigvee_{n=1}^{MSP} (P_n > H_{ln}) \tag{1}$$

$S_l(P_0)$ is a Boolean value that represents the shading of point P_0 for a given light source l. As stated above, $S_l(P_0)$ is equal to 0 if P is shaded, 1 otherwise. $\bigvee$ Symbol represents a logical OR. MSP is the maximum shadow propagation threshold, it represents the maximum size of a shadow for the given tile. Therefore, trying to find a obstacle after this threshold is a waste of time.

The Shading Algorithm uses three input data: The light/signal source position, the DUSM or DTM and the latitude of the location of the given model. This algorithm is observed to be an irregular time consuming algorithm: the complexity of the algorithm depends on sun position. The output of this algorithm is a shadow map that contains the shading state of each point that belongs to a tile.

Two paradigms are considered for implementing the Shading Algorithm: The “shadow receiver” and the “shadow emitter”.

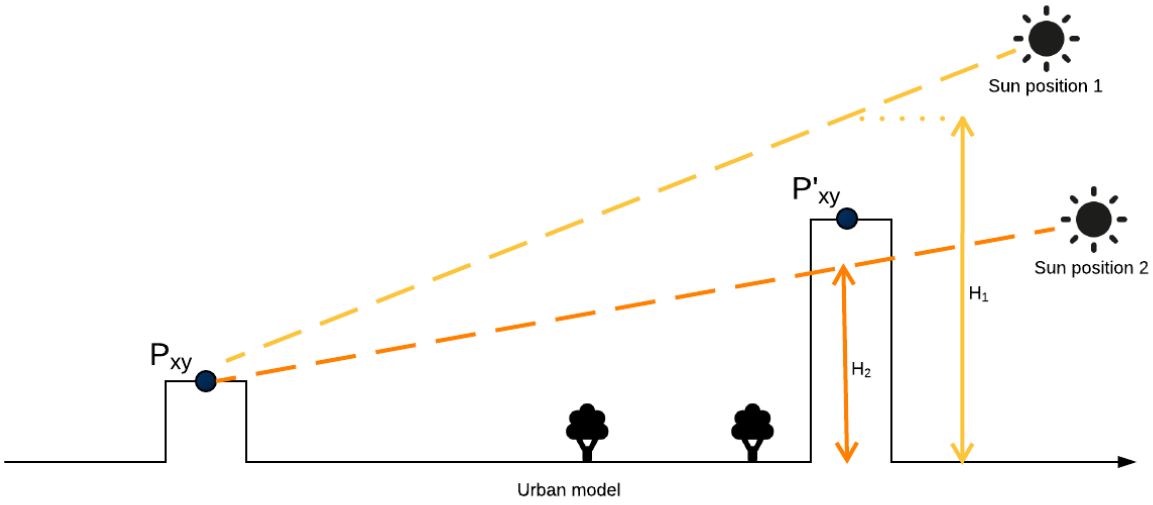

Figure 6 Principle of the Shading Algorithm.

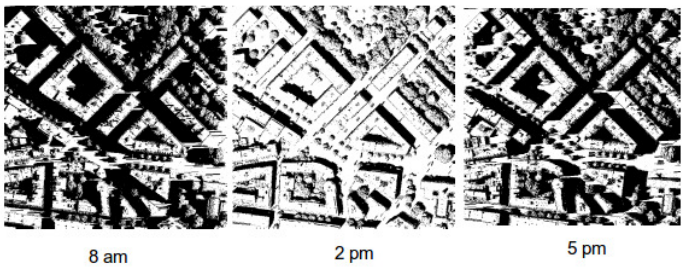

Figure 7 Calculated shadows for an area of Geneva at three times of the day.

3.1 Shadow receiver paradigm

The principle of the "shadow receiver" paradigm is as follows: for a given light source and a given point P, the algorithm proceeds in the direction of the light source and checks if the encountered points (buildings, trees, obstacles) hide P (see Figure 6). This algorithm searches for "barriers" that prevent P from being sunny [3]. This algorithm has been developed in two versions: point by point and matrix.

For each point P_{xy} belonging to the tile, the point by point version behaves as follows:

1. The direction of the light source beam is expressed as the light source beam vector $\begin{pmatrix} dx \\ dy \\ dz \end{pmatrix}$.

2. The neighbour P' of a given point P at given step i is given by:
   $P'_{xy} = \begin{pmatrix} x + round(i * dx) \\ y + round(i * dy) \end{pmatrix}$
   The height of P'_{xy} (i.e P'_z) is given by its value in the urban model. i is an iterator that starts from 1 and ends either when P' is outside the tile or $P_z + i * dz$ is greater than the highest point of the tile or P'_z is greater than $P_z + i * dz$ (i.e the neighbor is shading P).

3. A point is shaded if at least one of his neighbour is higher than $P_z + i * dz$

The matrix version of the Shading Algorithm has been introduced by [3], it behaves as follows: First, the algorithm computes the highest building's height in order to fix a threshold for the maximum light beam height. As a reminder, the light beam height at step i is denoted by $i * dz$. That is, when $i * dz$ (i.e the height of the light beam at step i) exceeds the threshold the algorithm stops. Then, as stated above, the algorithm will compute the light source beam direction vector (i.e $\begin{pmatrix} dx \\ dy \\ dz \end{pmatrix}$) directed to the light source. Afterwards, the algorithm behaves iteratively as follows. At each iteration, the light source beam vector components (i.e dx, dy, dz) are incremented by one and the neighbor matrix $N(i)$ of the current step i is generated. This matrix can be formalized as follows:

$$N(i) = \begin{bmatrix} f(P_{0,0}, i) & \cdots & f(P_{x,0}, i) \\ \vdots & \ddots & \vdots \\ f(P_{0,y}, i) & \cdots & f(P_{x,y}, i) \end{bmatrix}$$

Where $f(P, i)$ is a function that return the height of the next neighbor of $P_{x,y}$ at step i. Thus, for instance, the height of the next neighbor of P_{12} at step 3 is given by $N_{12}(3)$. Hence, a point P_{xy} is shaded whether at least one of its neighbors $N_{xy}(i)$ is higher than $P_z + i * dz$. The computation stops either dz reaches the threshold defined before or dx or dy are greater than the tile's dimension.

3.2 Shadow emitter paradigm

The Shadow Emitter paradigm simply consists of calculating, for a given light source and for each point P_{xy} belonging to the tile, the list of points that are shaded by P_{xy}: $LIST_{xy}$.

A point P, belonging to the tile, is sunny (for a given light/signal source) if it does not belong to any lists $LIST_{xy}$. In other words, P is not shaded by any other point P_{xy}.

Unlike the first alternative, the algorithm proposed in this section calculates the shaded points through a given point P_{xy} and not the shadows that fall on P_{xy} from nearby points.

This algorithm can be divided into two steps:

1. "shaded area identification": This step consists of identifying, for each point that belongs to a tile, what points are shaded by it (i.e identifying the $LIST_{xy}$) in function of a given light/signal source. For this purpose, the algorithm follows the light/signal source beam in the direction of the ground and checks if the encountered points (buildings, trees, obstacles) are shaded by P_{xy}, in this case they

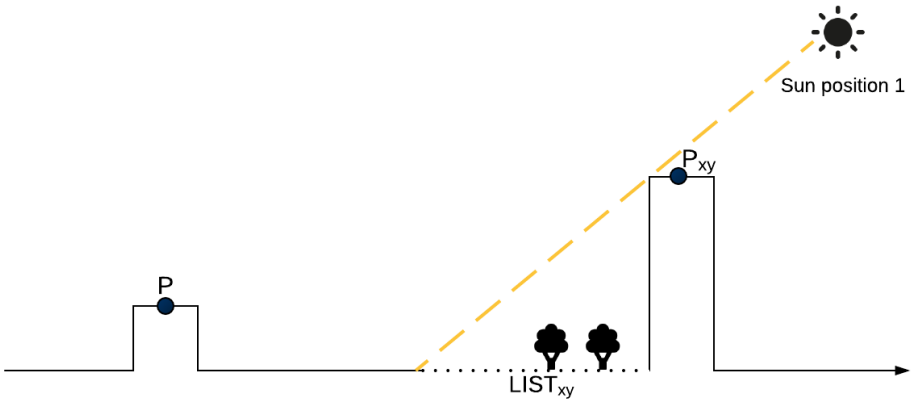

Figure 8 Principle of the shadow emitter paradigm.

are added to $LIST_{xy}$ (see Figure 8). The encountered points are computed iteratively and are given by:

$$P'_{xy} = \begin{pmatrix} x - round(i * dx) \\ y - round(i * dy) \end{pmatrix}$$

dx, dy and dz are the light source beam components. The height of P'_{xy} is given by its value in the urban model. i is an iterator that starts from 1 and ends either when P'_{xy} is outside the tile or if an encountered point is shaded. Therefore, P'_{xy} is shaded if it is lower than $P_z - i * dz$.

At this step, each point knows the points that it shades.

2. "merging" : This step consists to identify the points that belongs to any $LIST_{xy}$ and those that are shaded in order to construct the shadow map.

Figure 9 shows an example of the three steps defined above.

4 Solar radiation

The factor of solar radiation is a relevant information in case of thermal and photovoltaic, energy production. Hence, it becomes important to calculate an accurate estimation of the energetic production capability of roofs and facades. For this purpose, solar radiation maps are built from the input data specified in Section 2.1.

Solar radiation $SR_l(P)$ at a given point P depends on several factors, such as:

- Local weather;
- Latitude, slope and orientation of P (assuming that P is a roof point or a facade point);
- Sun position (denoted by l);
- How much shade falls on P from nearby trees and buildings.

As stated in Section 2.1, three types of radiation make up the total solar radiation: direct, reflected and diffuse. Formula 2 provides a simplified version:

$$SR_l(P) = I_{dir} * S_l(P) + I_{ref} + I_{dif} * (f(BPV(P), SVF(P)) + g(S_l(P))) \quad (2)$$

I_{dir}, I_{ref}, and I_{dif} are three constants which depend on weather data, latitude, slope and orientation of P. Functions f and g are respectively the circumsolar and the isotropic components of the diffuse radiation [4].

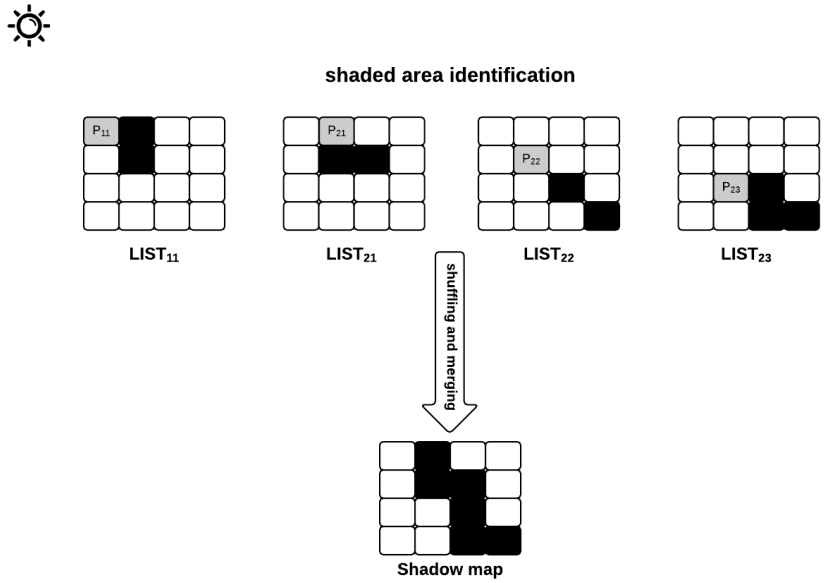

Figure 9 Example of steps in the shadow emitter paradigm. P_{11}, P_{21}, P_{22} **and** P_{23} **are higher than the ground, all the other points do not produce any shadow (i.e** $LIST_{xy}$**).**

For a given sun position l, $S_l(P)$ is equal to 0 if P is shaded, 1 otherwise. The Sky View Factor (SVF) will be explained in the following sub-section.

Formula 2) is divided into three parts:

1. Direct radiation ($I_{dir} * S_l(P)$) is directly proportional to the sun visibility;
2. Radiation reflected from buildings and ground is given by I_{ref};
3. Diffuse radiation ($I_{dif} * (f(BPV(P), SVF(P)) + g(S_l(P)))$) is derived from the sky visibility, direct sun visibility and the best possible view, which is described in Section 4.1.

The value of the direct solar radiation is zero if P (belonging to a roof) is shaded. The Shading Algorithm is used to determine whether P is shaded or not.

The reflected solar radiation represents the amount of light reflected by the environment such as buildings or the ground. It has a small impact on the total solar radiation.

The diffuse solar radiation depends on the ratio of the sky visible at P. When P is located among several obstacles, its sky visibility is limited (close to 0). On the other hand, if P is an open point (located outside urban areas for example) its sky visibility is close to 1. The sky visibility is modeled by the SVF that has a value between 0 and 1 (Figure 3). To clarify, the SVF is the percentage of sky that can be seen from a specific point.

The solar radiation calculation is based on three distinct processes:

- The first one is the SVF calculation which involves hundreds of shadow processes (Shading Algorithm) for an accurate estimation of the SVF. This step uses a DUSM and a sky model (See section 4.1) as input data.
- The second process is the hourly shading calculation which involves one shadow process per hour during a given time span (i.e a year, a month) for establishing the direct sun visibility. This step uses a DUSM and sun positions as input data.

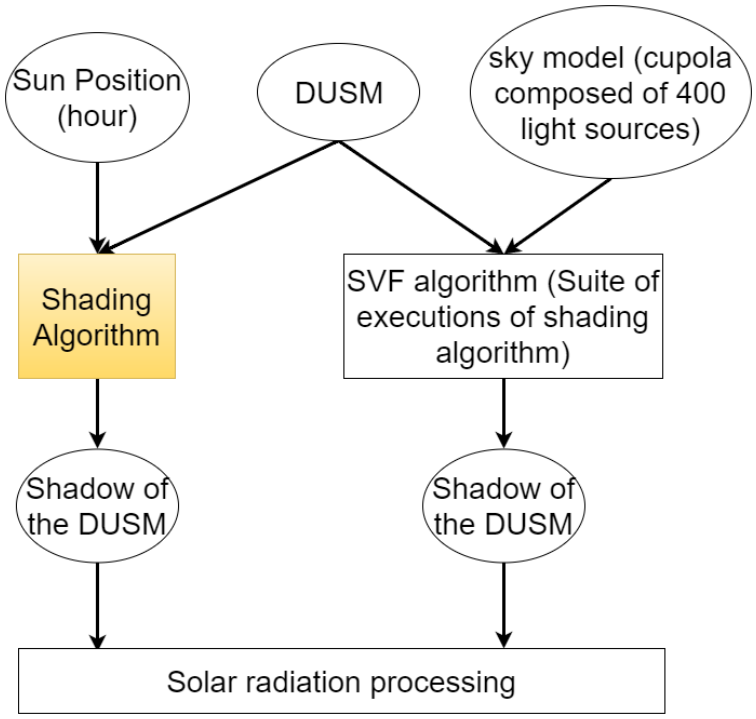

Figure 10 Functional diagram of the "Solar Energy Potential"; The colored square is the generic core engine of iCeBOUND which is involved in both GNSS and Solar Energy Potential.

- The results of these two processes are the main components of the diffuse radiation and the direct radiation respectively. Therefore, the last process consists in creating the solar map using the previous results.

Figure 10 shows the functional diagram of this use-case.

The next section will explain the computation of the SVF and the input data that is involved in its computation.

4.1 SVF

As stated in the previous section, the sky view factor is the percentage of sky that can be seen from a specific point. By definition, each point of the DUSM has a corresponding SVF but in our case, only the roofs' and the facades' points are points of interest. The SVF at point P (SVF(P)) is calculated as follows [5]:

- The sky is modeled by a cupola having different suns (light source) uniformly placed on the cupola;
- The Shading Algorithm is executed for each light source in the sky (cupola);
- SVF(P) is the average of all shadows calculated at point P.

The more light sources in the cupola, the more accurate is the SVF. In our case, the number of light sources in the sky is set to 400. Moreover, the SVF can be formalized by the following equation:

$$SVF(P) = \frac{1}{L} \sum_{l=1}^{L} S_l(P) \tag{3}$$

where l is the identifier of the light source, L is the number of light sources in the sky model. As a reminder, $S_l(P)$ denotes the shading state of the current point for the given sun (true = 0, false = 1).

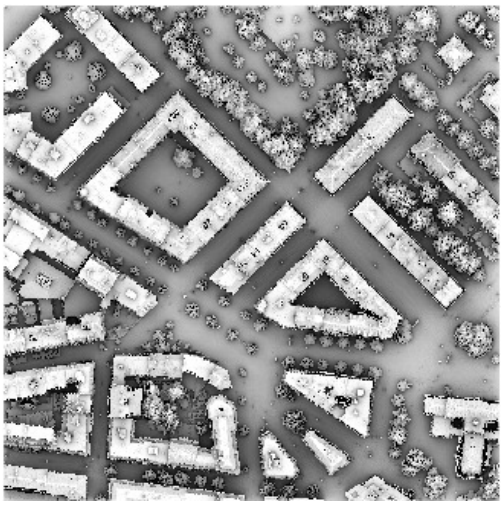

Figure 11 Output of the SVF algorithm.

Overall, the SVF is a suite of executions of the Shading Algorithm which is irregular and time consuming. Therefore, SVF needs resources with high performance computing capabilities.

In addition, it is important to notice that the SVF can not be larger in value than the best possible view (BPV) which can be computed as follows:

$$BPV(P) = 1 - \frac{\beta(P)}{180} \tag{4}$$

where $\beta(P)$ is the slope of the point P This value is used in the solar radiation formula (described in Section 4) to reduce the impact of the slope on the isotropic component of the diffuse radiation. This value concerns both roof and facade points.

The SVF process is done on every point of the targeted model and the output of this algorithm is a SVF map of the whole tile. Figure 11 shows an example of SVF map on an area of Geneva.

4.2 Facade

As introduced in Section 2.1, solar panels are most efficient when placed on roofs but, sometimes, facades can also be a good place to put solar panels when they are south oriented. Therefore, the same study of the solar energy potential should be carried out for building facades. For this purpose, the shadow receiver algorithm was extended to take into account the constraints added by the facades.

As long as the SVF is nothing but a series of shadow processes, it does not need to be modified.

In addition to the input data of the Shading Algorithm, the facade algorithm needs three additional files (matrices): the frontID, the frontOrient and the frontDTM matrix. The frontID matrix is used to know which points belong to a facade and to differentiate the facade with their unique ID. Each point of this matrix has an ID, this ID can be either:

- 0, that means the point does not belong to any facade.
- between 1 and 2^{32}, that gives the facade ID.

The values of this file are encoded in 32 bit integers.

The frontDTM matrix describes the level of the ground above the sea level on each point of the DUSM. Therefore, this file has the same resolution as the DUSM, which is 0.5 m. The values of this file are encoded in 32 bit integers. This file is used to know the height of each facade by matrix subtraction of the frontDTM from the DUSM.

The frontOrient matrix describes the orientation of each point that belongs to a facade of a given tile between 0 and 90°. The values of this file are encoded as 8 bit integers. This file is used to know if a light source is behind or in front of a given facade.

The slope matrix is no longer required because in our case the facades are considered to be vertical with a slope of 90°.

The outcome of these algorithms are, as mentioned in Section 2.1, linear files that contain, for each line, the absolute position of the point (X, Y, Z) and the result of the computation of the algorithm for it. That is, for instance, the outcome of the facade shadow process contains the absolute possition of the point and its shading state for a given light source.

5 PDOP

5.1 Calculation of visible satellites

Trimble Company [6] almanac provides information about global orbital parameters for all available satellites in the sky. Using these information and specific conversion algorithms, local coordinates that depend on the sight of seeing (in this case, GNSS receiver's position) are calculated from global orbital parameters. Hence, GNSS receiver's position in WGS84 and topocentric (altitude and azimuth) coordinates of each visible satellite from GNSS receiver's position are calculated (Yuen, 2009 [7] and Mevel, 2011 [8]). Using these parameters and DUSM as input data of our tool it is then possible to calculate visibility of each satellite according to the urban fabric. In addition, it is assumed that a line of sight is needed for satellite's signal to be visible to GNSS receiver. Synoptic view of figure 12 emphasizes this calculation process.

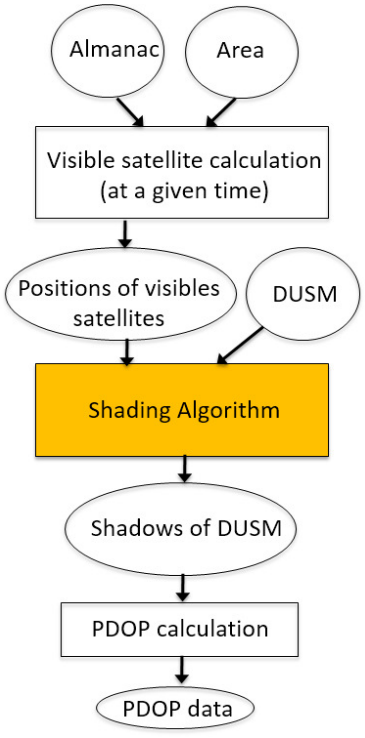

Figure 12 Workflow of the calculation of the PDOP.

5.2 Calculation of PDOP

The position dilution of precision (PDOP), also called spherical dilution of precision, relates to satellite geometry over the sky at a given time and location. PDOP value of a GNSS measurement should be as low as possible (ideally, lower than 5). Figure 5 highlights different satellites typologies in the sky that result in a good or bad PDOP.

The meaning of PDOP values calculated for a given time can be classified as follows:

- Ideal (lower than 1): highest possible confidence value, meeting the requirements of all applications,
- Excellent (1 to 2): accurate enough to meet requirements of most sensitive applications,
- Good (2 to 5): marks the minimum appropriate level for general applications,
- Moderate (5 to 10): could be used for certain applications. Nevertheless, a more open view of the sky is suggested,
- Fair (10 to 20): low confidence level. Hence, GNSS measurements should be rejected for the majority of applications,
- Poor (higher than 20): at this level measurements are considered totally inaccurate.

6 Parallelization

As said in previous sections, the Shading Algorithm is observed to be an irregular and time consuming algorithm. Moreover, both visible satellites calculation and SVF calculation execute this algorithm hundreds times to provide accurate information. To give an order of magnitude, computing 400 shadow process on a small tile (i.e 300 m^2) takes approximately 20 minutes.

The "shadow receiver" and "shadow emitter" paradigms have been parallelized. The computation of the shadow receiveralgorithm has been distributed on several machines using a master/slave approach with HTCondor whereas the shadow emitter algorithm has been parallelized using a MapReduce approach with Hadoop.

6.1 Master/slaves approach

The principle of this paradigm is well known in parallel and distributed computing. It consists of splitting the input data into small pieces and allocating them to different workers. Each worker processes jobs and sends results to the master machine, which performs a merge operation to build the final result. This model is called master/slave or Bag of Tasks (BoT).

In our case (SVF algorithm), input data is composed of:

- The 3D urban model of the city or town;
- The set of light sources (suns position) that models the sky (sky model).

The output data is the SVF matrix of the city or town.

The city is divided into several areas called tiles. Formally, a tile can be described as follow:

$$Tile = \begin{bmatrix} P_{0,0} & \cdots & P_{x,0} \\ \vdots & \ddots & \vdots \\ P_{0,y} & \cdots & P_{x,y} \end{bmatrix}$$

For each tile, the light sources belonging to the sky model are divided into sub-sets. Each sub-set is allocated to a given worker w. Each worker w executes the shading algorithm, as many times as there are "light sources" in its sub-set.

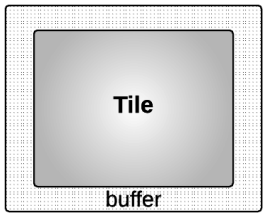

Figure 13 Buffer surrounding the tile is used to calculate shadings of the peripheral points of the tile.

In order to calculate if a point P is shaded or not, the algorithm needs the neighboring points of P. These points may not belong to the tile assigned to w. In this case, a data exchange between other workers (dealing with neighboring tiles) is necessary.

To avoid this data exchange overhead, the area transmitted to a worker w is larger than the tile processed by w (Figure 13). The buffer (strip) surrounding the tile is used to calculate shadings of the peripheral points of the tile. However, the shading of the points belonging to the buffer is calculated by neighboring workers.

The size of the buffer depends on the highest building in the tile: if the tile belongs to the city of Manhattan in New York, this strip should be wide enough to calculate the shadings of the points that are in the periphery of the tile.

The major disadvantage of this paradigm is related to scheduling and load balancing aspects. Since the shading algorithm is irregular, a small number of jobs (large number of light sources per job) could involve a significant imbalance among workers. While a larger number of jobs (fewer light sources per job) could resolve the imbalance problem but could involve a synchronisation overhead between workers and master.

HTCondor has been used as a cluster software manager [9]. The developed system follows these steps:

1. initiate, create and start an HTCondor cluster inside a cloud infrastructure.
2. create sub-tasks from the original task (Split the original data into small pieces).
3. submit the sub-tasks to the cluster.
4. retrieve the results and merge them.

An API has been developed in Python 2.7 in order to:

- Create compute resources on cloud providers such as Amazon Web Service [10], SWITCHEngine [11] and hepiacloud [12].
- Initialize, install and start a HTCondor cluster on a specific set of machines.
- Initialize and to create tasks from either a sky model or a “taskfile” which describes line per line the light source (azimuth and altitude) for a task (i.e each line is a task where both the altitude and the azimuth are specified).
- Perform lightweight monitoring of the running processes’ status (active, inactive, terminated).
- Retrieve results from the HTCondor master node and send them to the server that orchestrates the computation.

For these purposes, the following libraries have been used:

- Libcloud: As a cloud interoperability library [13]

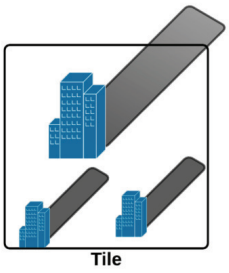

Figure 14 Shadows belonging to different sub-tiles.

- Fabric: As a remote system administration library [14]
- SciPy: As a tool for reading matlab matrix files
  (i.e DUSM, DTM, MNS, orient matrix, slope matrix) [15]
- boto: For accessing Amazon Simple Storage Service [16]

The installation procedure of these tools is described in appendices A.

6.2 MapReduce approach

Like the first alternative, this approach aims at dividing a big tasks into small tasks that can be performed in parallel. However, in this case, the MapReduce paradigm is used to parallelize the tasks on an Hadoop cluster. The "shadow emitter paradigm" is used as Shading Algorithm. The methodology is to divide the tile into sub-tiles and to submit them, in parallel to several Hadoop workers. Hence, it consists of calculating, for a given time of the day (a given sun position) and for each point P_{ij} belonging to the tile, the list of points that are shaded by P_{ij}: $LIST_{ij}$.

A point P, belonging to the tile, is sunny (at a given time: a given sun position) if it does not belong to any lists $LIST_{ij}$. In other words, P is not shaded by any other point P_{ij}.

Unlike the first alternative, the algorithm proposed in this section calculates the shaded points through a given point P and not the shadows that fall on P from nearby points.

A parallel version of this algorithm could be written as follows: tiles are split into sub-tiles. Each worker W processes one sub-tile ST_w. Three steps can be executed in parallel:

- Each worker w calculates the set of lists $LIST_{ij}$ of all points P_{ij} belonging to the sub-tile ST_w. More concretely, each worker processes the shadows generated by all the points belonging to its sub-tile ST_w. This step is called shadow mapping. It is worth reminding here that points belonging to $LIST_{ij}$ could belong to different sub-tiles (Figure 14).
- Each worker w sends the result of its shadow-mapping step to the other workers: a worker w informs the other workers that the points belonging to their sub-tiles are shaded by one of the points belonging to ST_w. This step is called the shuffling step.
- The last step consists of merging the result and identifying the points which are shaded and those which are sunny: if a worker w does not receive any information regarding a given point P belonging to ST_w, this means that P is sunny. This step is the reducing step.

The shuffling and reducing steps can start executing even though the first shadow-mapping step is not yet finished. This way of expressing parallelism is very similar to the MapReduce paradigm.

Amazon Elastic MapReduce [17] with Apache Hadoop 2.0 [18] has been used to process the MapReduce batches. The Hadoop framework implies some programming constraints:

1. The input format must be either readable by a Hadoop standard input reader or readable by a home-made Hadoop input reader.

2. The data type transferred between mapping, shuffling and reducing step must be well programmatically defined.

3. The results from the reducing step are written in chunks that have to be merged afterward to complete the process.

By default, Hadoop can not read the GeoTIFF format. Therefore, the DUSM or the DTM is converted into the standard Hadoop "FileInputFormat". In this format, the model is described line per line where each line corresponds to a point in the DUSM. For instance, the first line corresponds to the top-left element of the DUSM matrix and the second line corresponds to the next element to the right and so on.

The position of the light sources, the number of reducers, the size of the input split, the number of light sources to process (if it differs from the number of light sources' position) are some inputs of a shadow emitter batch and are stored in the Hadoop File System (HDFS) to be shared between the nodes.

During the execution of the Hadoop batch the data transfer proceeds as follows:

1. The input reader reads the DUSM matrix to produce entries: for the mapping step in key value pairs format of type: Long $\rightarrow$ String.

2. The mapping step executes the "areas identification" process to produce results for the shuffling step of key value pairs with the format: Tuple<Long, Long> $\rightarrow$ Boolean.

3. The shuffling step produces results for the reducing step in the form of key value pairs with the format: Tuple<Long, Long> $\rightarrow$ List<Boolean>.

4. The reducing step will merge the results and write to HDFS key value pairs of type: Long $\rightarrow$ Double.

The output of a Amazon Elastic Map Reduce (EMR) Hadoop batch are written in Amazon S3. These results are in a linear format like the input file, therefore they need to be merged in order to build the expected result (i.e satellites visibility map, shadow map, SVF map). For this purpose, a MapReduce results reader has been developed to merge the results produced by a "shadow emitter" batch. The purpose of this reader is to read the files produced by EMR in Amazon S3 and to create a GeoTIFF file that contains the values produced by Hadoop.

6.3 Experimental results

In order to compare the different implementations of the Shading Algorithm, a comparative study has been done between the shadow receiver algorithm and the shadow emitter. This study targets the performances and costs of these algorithms for the computation of the SVF. The goal of this study is to figure out which algorithm is: the *cheapest* and the *fastest*.

The computing resources used to experiment the algorithms presented in section 2 and 3 are based on Cloud infrastructures: Amazon Web Services (AWS), SWITCHEngines [11] (SWe) and Amazon Elastic MapReduce [17] (EMR). These infrastructures were chosen on the basis of the ease of availability in the case of AWS and the possibility to obtain a customized flavors/solutions in the case of SWe. The computing presented here uses a single-thread application and requires a large amount of RAM for efficiency.

The input tile is a 3.4 km by 3.4 km urban area of G, Switzerland. The resolution is 0.5 m: one pixel represents a square of 05 * 0.5 m. The sky model contains 400 light sources.

The performance of the two paradigms, shadow receiver and shadow emitter, is measured by the running time and the overall cost to perform the calculation on the given urban tile. The running time is simply the wall-time needed to complete the calculation and is measured as a function of the number of virtual CPUs (vCPUs) assigned to the job. The cost of performing the calculation is obtained directly from the price that Amazon charged.

Parameter	AWS	SWe
Node flavor	r3.large	r3.medium
Price per hour	$ 0.166	free
vCPU per VM	1	1
RAM per VM	20 GB	10 GB

Table 1 Specifications of VMs on AWS and SWe.

The shadow receiver computations are performed on AWS and SWe with the specifications given in Table 1. The submission and synchronization of the computations are handled by HTCondor [9] and libCloud [13]. libCloud is employed in order to avoid problems with proprietary cloud APIs. libCloud is an open source API that guarantee portability and interoperability among different cloud infrastructures (write once, run anywhere).

Figure 15 shows the wall-time of the shadow receiver SVF calculation of the urban tile as a function of vCPUs on both AWS and SWe. It is apparent that the performance of the shadow receiver calculation is similar on the AWS and SWe infrastructures using VMs with the specifications given in Table 1. It can be seen on Figure 15 that the wall-time of a calculation does not always decrease with increasing number of vCPUs. This can be explained by the randomness of the execution times of the tasks in the BoT. The number of tasks is equal to the number of vCPUs but some tasks finish earlier than others and the whole calculation must wait for the slowest task. The time taken by the algorithm to calculate whether a point is shadowed or not is not constant. In general, shadow calculations take longer for sun positions close to the horizon.

This anomaly of increased wall-time could be mitigated by adjusting the sun positions for the SVF. It can be postulated that the calculations of solar energy potential for sun positions close to the horizons are less interesting as the sunlight is more diffuse. Therefore the number of sun positions closer to the horizon for the SVF calculation may be reduced.

Another strategy to reduce the wall-time anomaly is to decrease the size and increase the number of jobs (small number of light sources per job). This will result in a larger number of shorter duration tasks that have a smaller deviation in calculation times. A practical limit on the number of jobs will be set by the overhead of the HTCondor software that synchronizes the calculations.

As stated earlier, the imbalance (master/slave version) is due to the fact that the execution time of shading algorithm, for a given area, depends on "sun position". In other words, the algorithm shows an irregular behavior. The experimental measurements of the master/slave version concern a static distribution of tasks among instances. To address load imbalance, several scheduling techniques can be used to resolve this problem. For instance, dynamic loop scheduling techniques (DLS) [19] such as Fixed Size Chunking [20], Weighted Factoring [21] or Adaptive Factoring [22] have been proven to optimize the completion time of parallel loops in such applications. The goal of the project is not to study DLS strategies, but to show that our DSS prototype can guarantee high performance throughput based on an "inexpensive" cloud infrastructure.

The cost of performing the SVF calculation on AWS (SWe is free) is shown on the right-hand Y-axis of Figure 15. The cost curve is approximately flat at around $ 20 in this case, not varying appreciably over 20 to 40 vCPUs.

The shadow emitter SVF calculations for the urban tile are performed on Amazon Elastic MapReduce (EMR). The measurements have been performed on EMR with the VM parameters listed in Table 2. Figure 16 shows the wall-time results and associated price charged by Amazon obtained in EMR.

In Figure 16 the wall-time of the SVF calculation clearly decreases as the number of vCPUs increases. In comparison to Figure 15, the wall-times of the shadow emitter calculation on EMR are generally but not appreciably lower. The clear difference between Figures 15 and 16 is the price charged by Amazon for the calculation. In general, the cost of the shadow emitter calculation on EMR is one third of the cost compared to running the process-driven calculation on AWS.

The price vs. number of vCPUs in both Figures 15 and 16 shows a sawtooth pattern. This pattern is an artefact of the Amazon pricing model which charges by whole hours or a portion thereof. For examples, a

Parameter	Value
Master node flavor	m1.medium
Price per hour	0.0331 $
EMR increase	0.022 $
Compute node flavor	m3.xlarge
Price per hour	0.2660 $
EMR increase	0.07 $
vCPU per compute node	4

Table 2 VM parameters on EMR.

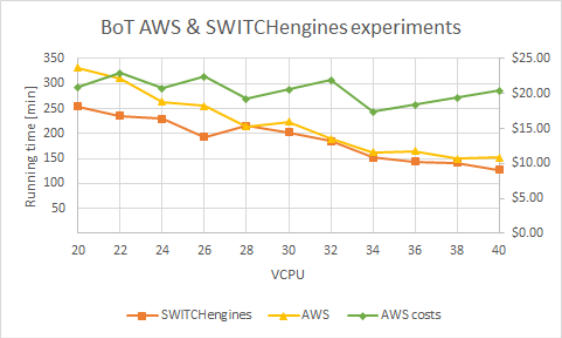

Figure 15 BoT experimental results; Orange and yellow curves show the computation time of the BoT solution in AWS and SWe respectively. The green curve shows the cost of this solution in AWS.

calculation that takes 2 hours and 1 minute is charged for 3 hours. Therefore the cost of using more vCPUs will increase until the wall-time drops to the next lowest full hour. As a result, in Figure 16, it can be seen that 80 vCPUs gives a wall-time of approx 1 hour. Any increase in vCPUs will not decrease but will increase the cost as the extra vCPUs will be charged for the full hour.

7 Conclusion

Two decision support systems (DSS) have been developed in the iCeBOUND project:

- The first DSS is used to calculate the solar energy potential of surfaces in urban landscapes. The system is assumed to identify good candidate roofs for installing solar panels. The goal is to quantify the solar power a roof could generate by taking into account local weather, roofs orientation and how much shade falls on it from nearby trees and buildings.
- The second DSS is used to calculate the number and position of available satellites in the sky at a given time and a given position: Global Navigation Satellite System (GNSS). The main goal here is to optimize human resources and equipment needs for topographic measurements to be done on urban areas.

The two developed DSSs are based on the same CPU and memory intensive shadow process algorithm Shading Algorithm. In this report, we propose two paradigms to describe this algorithm. The first is based on a conventional parallelization. The second paradigm is based on an optimized data distribution. The two paradigms are implemented on cloud computing infrastructures. The report compares the two methods on the basis of two criteria: performance and deployment cost.

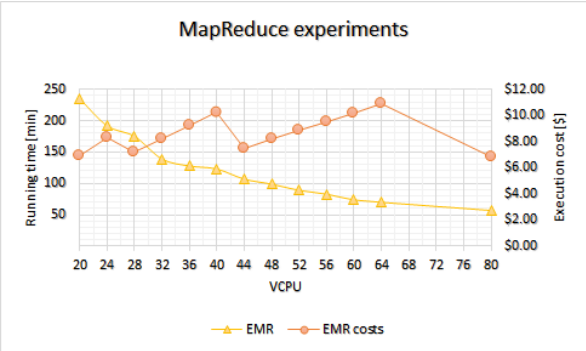

Figure 16 MapReduce experimental results; Orange (resp. red) curve shows the running times execution cost) of the MapReduce solution on EMR.

The source code of the whole project is available on these GitHub links:https://githepia.hesge.ch/lsds/iCeBOUND
The solar energy DSS is available on this URL link: http://lsds.hesge.ch/icebound
The GNSS DSS is available on this URL link: https://hepiageo.hesge.ch/visibilitegnss/

References

[1] T. Santos, N. Gomes, S. Freire, M. Brito, L. Santos, and Tenedorio, "Applications of solar mapping in the urban environment," *Applied Geography*, vol. 51, pp. 48 – 57, 2014.

[2] "jai-imageio." https://github.com/jai-imageio/jai-imageio-core.

[3] C. Ratti, S. Di Sabatino, and R. Britter, "Urban texture analysis with image processing techniques: winds and dispersion," *Theoretical and Applied Climatology*, vol. 84, no. 1, pp. 77–90, 2005.

[4] M. Iqbal, *An Introduction to Solar Radiation. Chapter 4 - Extraterrestrial solar irradiation, pages 59 - 84*. Academic Press, 1983.

[5] P. Redweik, C. Catita, and M. Brito, "Solar energy potential on roofs and facades in an urban landscape," *Solar Energy*, vol. 97, pp. 332 – 341, 2013.

[6] "Trimble company." http://www.trimble.com/.

[7] M. F. Yuen, "Dilution of precision (dop) calculation for mission planning purposes," 2009.

[8] B. Mevel, "Cartographie et potentiel de lutilisation du gnss sur le canton de genve dans le contexte de relev de rseaux eau, gaz et lectricit au sein de lentreprise sig," 2011.

[9] "HTCondor." https://research.cs.wisc.edu/htcondor/.

[10] "Amazon Elastic Compute Cloud." https://aws.amazon.com/fr/ec2/.

[11] "SWITCHengines." https://www.switch.ch/engines/.

[12] "hepiacloud." http://lsds.hesge.ch/hepiacloud/.

[13] "libCloud." https://libcloud.apache.org/.

[14] "fabric." http://www.fabfile.org/.

[15] "Scipy." http://www.scipy.org/.

[16] "boto." https://github.com/boto/boto.

[17] "Elastic MapReduce." https://aws.amazon.com/elasticmapreduce/.

[18] "hadoop." http://hadoop.apache.org/.

[19] C. R. L. Banicescu, Ioana, "Addressing the stochastic nature of scientific computations via dynamic loop scheduling.," *ETNA. Electronic Transactions on Numerical Analysis [electronic only]*, vol. 21, pp. 66–80, 2005.

[20] C. P. Kruskal and A. Weiss, "Allocating independent subtasks on parallel processors," *IEEE Transactions on Software Engineering*, vol. SE-11, pp. 1001–1016, Oct 1985.

[21] S. F. Hummel, J. Schmidt, R. N. Uma, and J. Wein, "Load-sharing in heterogeneous systems via weighted factoring," in *Proceedings of the Eighth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '96, (New York, NY, USA), pp. 318–328, ACM, 1996.

[22] I. Banicescu and Z. L. A. Factoring, "A dynamic scheduling method tuned to the rate ofweight changes.," *In High Performance Computing Symposium (HPC 2000)*, pp. 122–129, 200.

A Installation guide of MeasureAPI environment

```
================================================================================
Documentation:
Installation guide of MeasureAPI environment  used to dothe iCeBOUND measures.
3 steps :
    1 - Setup the "measures folder" (Copying files like Launcher
        app, config/ folder, fabfile etc.)

    2 - Setup the env. of Launcher (Installing boto, fabric, libcloud

    3 - Setup your AWS account to be able to launch a cluster of machines
================================================================================

1 ) Folder architecture
    The folder where you want to do the measures MUST contains :
    1 - Launcher (the application)              //Latest version available in the SVN
    2 - fabfile.py (fabric configuration file)  //Latest version available in the SVN
    3 - config/  (condor configuration folder)  //available in the SVN
        condorCntrl.conf
        condorWrkr.conf

2 ) Environment
    Launcher use boto, fabric and libcloud libraries from python package (pip)
    Ensure you have python 2.7 and NOT python 3.x (it is not compatible)

    * If you don't have python2.7 :
      sudo apt-get install python2.7

    * To install these libraries you must have pip installed.
      Otherwise you should do execute these commands:
      sudo apt-get update
      sudo apt-get install python-pip
      sudo pip install --upgrade pip
      * You should now have the latest version of pip

    [boto]     ->
        sudo pip install boto

    [libcloud] ->
        sudo pip install apache-libcloud

    [fabric]   ->
        sudo apt-get install fabric

    [Scientific python (to read math lab files)] ->
        sudo apt-get install python-scipy

    tests :
    user@uservm: ~/python
    Python 2.7.6 (default, Mar 22 2014, 22:59:56)
    [GCC 4.8.2] on linux
    Type "help", "copyright", "credits" or "license" for more information.
    >>> import boto
    >>> import fabric
    >>> import libcloud

    * At this step your environment should be completely installed.

3 ) AWS Account
    Launcher use AWS to launch a cluster of machines, AWS should be able to
    create a cluster of communicating machines.

    In order to do that, you need a well parameterized VPC (virtual private cloud)

```

```
1 - Log into your AWS account

2 - Go to the VPC service

3 - Click on "Your VPCs" on the left side bar and then "Create VPC"
    3.1 - Fill the fields
      Name : The VPC name ex measurevpc
      CIDR block is the network of the VPC, I recommend 10.0.0.0/16
      keep tenancy "default")
    3.2 - Edit the DNS Hostname / Resolution by Right clicking on your VPC
          and then click on "Edit DNS Hostname". Click on "yes" and save.
          Do the same operation for "Edit DNS Resolution"
    * You have now an empty VPC, now you need to define a subnet in your vpc

4 - Click on "Subnets" on the left side bar and then "Create Subnet"
    4.1 - Fill the fields
      Name : The subnet name e.g "measuresubnet"
      VPC  : Select your VPC (defined above)
      Availability zone : No Preference
      CIDR block is a subnet of your VPC network, I recommend 10.0.1.0/24
    4.2 - Allow automatic assign Public IP
      Right click on your subnet then click on
        "Modify Auto-Assign Public IP" -> "Enable ..." -> Save
    * You now have a VPC with a subnet, now you need a internet gateway
      to have an internet access

5 - Click on "Internet Gateways" on the left side bar and then
    "Create Internet Gateway"
    5.1 - Fill the name field with the name of your internet gateway,
          e.g "measureigw"
    5.2 - Attached it with your VPC by right clicking on your internet gateway
          and then click on "Attach to VPC", finally select your VPC "measurevpc"
    * You now have a VPC with a internet gateway associated, you still need a route
      table

6 - Click on "Route Tables" on the left side bar and the "Create Route Table"
    6.1 - Fill the name field, e.g "measurert" and select your VPC (measurevpc)
    6.2 - Edit your route table to define the default outbound rule
        6.2.1 - Click on your route table "measurert" and select the "Routes" tab
                below. Then, Edit it, add another route and put :
                "0.0.0.0/0" in Destination.
                Select your internet gateway in Target.
                and then save.
                * in other words : everything doesn't go to our subnet goes outside
    6.3 - Associate the route table with your subnet
        6.3.1 - Click on the "Subnet associations" tab and then click on Edit.
                Finally, associate your subnet "measuresubnet" and click on Save.
    * You now have a well parameterized VPC without Security group, and you need
      one.

7 - Click on "Security groups" under "Security" on the left side bar and then
    click on "Create Security Group"
    7.1 - Fill the fields
         Name : The name of your security group, e.g "measuresecgrp"
         Groupname : AWS does it for you
         Description : Give a description, e.g "measure security group with all traffic allowed"
         VPC : Select your VPC (measurevpc)

    7.2 - Edit the rules.
        7.2.1 - Click on your VPC and select the "Inbound Rules" tab below and then
                click on Edit.
                On "Type" select : "All Traffic"
                On "Source" type : Any where 0.0.0.0/0
                * In other words : Allow all traffic coming from outside (block nothing)
                And save !
```

```
            7.2.2 - The outbound rules are automaticaly set to "All traffic allowed"

    * Now we have a completely well defined VPC with a security group associated.

4 ) Reminder
    - Don't forget to have a keypair on your desktop
    - Due to a strange issue, sometimes you don't have the right to create
      a new bucket (with boto) so I recommend to create a bucket in S3 "by hand".

5 ) With all these steps completed you normally can start measure with command like :

      sudo ./Launcher -d dusm.tif -r 0.5 --cred credentials --bucket icebound-new -i keypair.pem
      --jar ShadowProcess.jar --results outputfolders -n 2 --jobname testmeasure
      --taskfile taskfile --secgroup measuresecgrp --subnet measuresubnet --flavor r3.large --refresh 1

      Tips : If you work with the large dusm, don't forget to use the --nopush option
             (the software will seek an already uploaded dusm named "dusm.tif" inside the --bucket and will use it) .

6 ) Describing your tasks with the taskfile
    In order to describe how many tasks (and how many shadow processing by task) we have to compute,
    it is necessary to write a taskfile.
    It is a very simple format. The taskfile is a file where each line correspond to a task scheduled by htCondor.
    A line is composed by a list of altitude (separated by space),
    a list of azimuths(separated by space) and these lists are separated by ';'
    Alt1 Alt2 AltN;Az1 Az2 AzN
    Example :
    45 45 45 45;10 20 30 40
    35 35;177 187
    55;12
    This taskfile is composed by 3 tasks.
        The first one will compute the shadow process for the pairs (altitudes azimuts) (45,10) (45,20) (45,30) (45,40)
        The second one will compute the shadow process for the pairs (altitudes azimuts) (35,177) (35,187)
        The third one will compute the shadow process for the pairs (altitudes azimuts) (55,12)
    It's recommended to not put empty lines in this file

=> Now, we can start measures.

In addition, here is the Launcher helper
*******************************************************************************

usage: Launcher [-h] -d D -r R [--subnet [subnet-name]] --cred <key:secret>
                --bucket name -i key [--keyname [keyname] [-n [N]] --jar JAR
                [--jobname [name]] [--results [local]] [--flavor flavor]
                --taskfile jobfile --secgroup secgrp [--nopush] --refresh
                REFRESH

Distributed Shadow Process

optional arguments:
  -h, --help            show this help message and exit
  -d D                  location of the digital surface urban model file
                        (local)
  -r R                  resolution of the digital urban surface model
  --subnet [subnet-name]
                        specify the subnet name of a VPC, the instance will be
                        powered up inside the VPC and in the specific subnet.
                        If you dont specify this parameter, your default VPC
                        will be used
  --cred <key:secret>   credentials of your S3 account
  --bucket name         bucket used
  -i key                private key file location
  --keyname [keyname]   key name, only required if filename differs from
                        keyname
  -n [N]                number of instances
  --jar JAR             Jar to deploy
```

```
194     --jobname [name]      Jobname
195     --results [local]     output results folder, default=current directory
196     --flavor flavor       flavor used for amazon, default=t2.micro
197     --taskfile jobfile    The file which describes the job
198     --secgroup secgrp     security group of the HTCondor cluster, MUST be
199                           associated with the VPC where the machine will be
200                           powered up
201     --nopush              you can active this option to not upload the dusm file
202                           and use an already uploaded one in your bucket
203     --refresh REFRESH     refreshing time of htcondor in seconds (every X
204                           seconds checks the jobs queue of htcondor)
205
206   ********************************************************************************
```