Heuristic Geo Query Decomposition and Orchestration in a SOA

Michael Ovonibi, Peter Baumann
Center for Advanced Systems Engineering (CASE)
Jacobs University
Bremen, Germany
{m.ownibi, p.baumann}@jacobs-university.de

ABSTRACT
SOA is often used today for architectures distributing geo-processing computation tasks. These tasks usually are both data and computationally expensive. A fundamental issue is finding an automated, efficient task distribution and coordination among different services. We propose a language-based approach to this issue. First, the distributed geo-processing problem is written up as a declarative query using the Open Geospatial Consortium (OGC) Web Coverage Processing Service (WCPS) language standard. Second, translate the query is translated into a graph on which a rule-based graph decomposition algorithms generates a task distribution among a subset of the service hosts available. A heuristic cost function, preliminarily based on the data output sizes of each query operator, is used to partition the query and determine the execution hosts of sub-queries. The result is re-assembled into a recursively nested query which also performs coordination of its distributed evaluation.

Categories and Subject Descriptors
D.2.10 [Software]: Design; H.2.8 [Database Management]: Database applications— Spatial databases and GIS

General Terms
Design, Experimentation

Keywords
Geo-processing; WCPS; Graph partitioning; Service orchestration

1. INTRODUCTION
SOA is often used today for web service architectures where distributed geo-processing tasks are required. Distributed geo-processing is necessitated by the distributed nature, size, and complex evaluation algorithms of geo-data. Such distributed processes, in the future, often will be embedded into some overall service which hides the complex data evaluation tasks behind simple, easy-to-use services. Geo-services are standardized by the Open Geospatial Consortium (OGC). The Web Coverage Processing Service (WCPS)[5], one of the standard geo-services, is considered in this paper.

WCPS specifies the syntax and semantics of a query language which allows for server-side retrieval and processing of multi-dimensional geospatial coverages [5]. Coverages, a type of geospatial data, encompass any spatio-temporally extended phenomenon [17] [23]. As currently overarching query languages in this generality are not sufficiently understood, WCPS focuses on multi-dimensional raster data. Examples include satellite imagery, thematic maps, digital elevation models, and climate model data. Coverages sizes are typically large. WCPS queries are given as expressions composed from coverage-centric primitives. As such, the WCPS language can be understood as a declarative notation for a coverage processing workflow.

The rest of the paper starts with a description of WCPS in Section 2, followed by our approach for decomposing and orchestrating geo raster queries in Section 3. A sample implementation is presented in Section 4 and related work is discussed in Section 5. In Section 6, conclusion and outlook are given.

2. WCPS
Simply put, a raster coverage in WCPS consists of a multi-dimensional array of some dimensionality, some extent (its spatio-temporal domain), and over some cell (pixel, voxel) type (its range type). WCPS defines a declarative, set-oriented query language on such raster coverages. Shaped in the style of XQuery and the tradition of SQL, a WCPS query contains an iterator definition, a processing expression, and optionally a filter expression. The overall query structure is as follows:

\[
\text{for } \mathbf{c}_i \in \langle \mathbf{C}_1, \ldots, \mathbf{C}_m \rangle, \ldots, \mathbf{c}_n \in \langle \mathbf{C}_0, \ldots, \mathbf{C}_{n-1} \rangle \text{ return processingExpr}(\mathbf{c}_1, \ldots, \mathbf{c}_n) \text{ where booleanExpr}(\mathbf{c}_1, \ldots, \mathbf{c}_n)
\]

This can be seen as a nested loop where each \( \mathbf{c}_i \) is bound to the \( \mathbf{C}_{i,j} \) coverages in turn. For each variable binding, the where clause is evaluated first. Only if the Boolean expression evaluates to true a result element for the response list will be generated. If this is the case, the processing expression is evaluated on the current variable assignment resulting in either a coverage, or scalar summary data, or coverage metadata. The operators provided for such expressions can be grouped as shown below.

- Geometric operations extract some subset of cells which together again form a coverage. Trimming retrieves a sub-coverage whose dimensionality remains unchanged. Slicing delivers a cutout with reduced dimensionality.
- Induced operations apply cell type operations to all cells in coverage simultaneously. This includes arithmetic, trigonometric and logical operations, and casting.
- Coverage summarization includes aggregation operations like count, average min, max, someth and all() quantifiers.
- All of the above functions actually represent convenience operators which can be reduced to a coverage constructor, an aggregator, or a combination thereof.
- Scaling and reprojection (warping an image into another coordinate reference system) constitute non-atomic function.
- Data format encoding specifies how coverage-valued results are to be shipped back to the client. The list of such encodings includes formats like TIFF, NetCDF, or SEG-Y.

Let us look at an example. Given three coverages A, B, and C, the following coverage expression returns the cell-wise addition of coverages A and B, incremented by the maximum pixel value found in C.:

\[
\text{result coverage is shipped back in TIFF format.}
\]

for a in (A),
   b in (B),
   c in (C)
return
   encode(a + b + max(c), "tiff")

Formally, a WCPS query can be modeled as an operator tree \( QT = (V, E) \) where \( V = \{v_1, ..., v_m\} \) is a set of nodes \( v_i \) which represents coverage processing operators \( E = \{e_1, e_2, ..., e_n\} \). \( e_i = (v_i, v_j) \) is a set of edges such that each edge \( e_i \rightarrow \) represents the dependence of operator \( v_i \) on \( v_j \). Figure 1 shows the tree for the above sample query. It consists of different operators processing coverages (shown in gray).

![WCPS Query Tree](image)

3. DISTRIBUTED WCPS FRAMEWORK

Due to the fact that the semantics of the WCPS service request is known both to the client and servers, automatic service dispatching, chaining and optimizing is possible. Therefore, we propose a framework where severs can collaboratively share data, load, and applications. This consists of several WCPS servers together with a distributed resource-aware registry where servers can publish their processing capacity, geo-applications and coverage’s metadata. A WCPS server, having received a request, can automatically extract portions that are best resolved locally, distribute other requested parts to other suitable servers, re-collect results and package them. Geo-applications servers can therefore, dynamically speed up and scale up their capacity in response to the cost and complexity of a geo-processing request. Although this is similar to grid computing, it differs from grid in terms because it is focused on geo-applications, and the tasks’ running times are typically smaller than in grid computing.

Therefore, given a query with query tree \( QT \), a set of homogenous WCPS servers, and a function which associates a coverage processing operator with its data output size, we find an optimal distribution of the execution of the query unto the different service available such that the data transfer between different hosts is minimized. This reduces the network cost of transferring the typically large sized coverages, and the processing cost at every subsequent host. Using the operators’ host information, the query is decomposed and orchestrated.

3.1 Host Assignment

We first look into the initial assignment of query operator nodes to hosts. Let the following be given.

- A query tree \( QT \) with a set of leaf nodes \( L \subseteq V \), which are the coverage reader operator (which reads coverages from disk).
- A set of homogenous computing hosts \( H = \{h_i \mid i \in N\} \) where each host \( h_i \) represents a WCPS server. To minimize data transfer, we restrict \( H \) to the host on which the query is received (referred to as master host \( h_m \)), and the hosts on which the coverages are located i.e. leaf operators’ hosts.
- A set of two-element set (consisting of an operator node and its associated execution host) which is denoted as Node-Host configuration \( NH = \{vh = (v \in V, h \in H) \mid \forall v \in V\} \). Note that the hosts associated with leaf operator nodes are fixed, and hence represent a restriction on the Node-Host configuration.
- A function host: \( V \rightarrow H \) (returns the host assigned to a node.)

The cost of distributed execution of the query tree \( QT \) with a Node-Host configuration \( NH \) is given by the function \( cost(QT, NH) \). The function is defined below as:

\[
\text{cost}(QT, NH) = \sum_{e \in E} \text{transport}(e)
\]

\( transport: E \rightarrow \mathbb{R} \) i.e. the cost of transport of data on an edge \( e \in E \). Its value is obtained as given below:

\[
\forall e = (v_i, v_j) \in E:
\]

\[
\text{host}(v_i) = \text{host}(v_j) \Rightarrow \text{transport}(e) = 0
\]

\[
\text{host}(v_i) \neq \text{host}(v_j) \Rightarrow \text{transport}(e) = \text{output-size}(v_j)
\]

We aim to derive a \( NH \) for \( QT \) which minimizes cost. In other words, we attempt minimizing the data transport between different hosts where sub-queries are executed. Mathematically,

\[
NH_i : cost(QT, NH_i) = \min \{\text{cost}(QT, NH) \mid i \text{ is the node-host configuration identifier}\}
\]

Deriving the minimum cost Node-Host configuration is a NP-hard problem; hence, a heuristics-based procedure for obtaining this is proposed. Before going into the details of the proposed procedure, we define the functions below

\[
\text{parent}(v): \text{The parent node of a node } v
\]

\[
\text{child}(v, i): \text{The child node (identified by } i \text{) of node } v.
\]

\[
\text{input-size}(v): \text{The total data input size of an operator node}
\]

Heuristic rules used to assign nodes to their hosts are given below.
The WCPS query model is extended with the ability to partitioning and slicing the orchestration
of the various services needed for the processing of the WCPS query. The WCPS server acts as the orchestration engine using the orchestration information specified in the orchestration nodes. Each sub-graph is translated to a sub-query, and the hierarchical composite service derived from the orchestration schema graph is executed. This implies that, starting from the host associated with the root of the WCPS query (the master host $h_m$), the WCPS server associated with every on every parent sub-graph (sub-query) recursively invokes the WCPS services on hosts associated with the children sub-graphs. To reflect the orchestration operations, we suggest two changes to the WCPS language on the conceptual level. The first one is to allow nested queries, that is: a query in place of the coverage list in the for clause. This will allow us to separate processing steps which we want to assign to particular hosts. The assignment of a query to a particular host is done with the second extension we propose. It adds a clause to a sub-query which references the host to which it should be sent for evaluation. Such a splitting makes sense if, in our previous example in Section 2, coverages $A$ and $B$ reside on the same node whereas $C$ is stored on a separate node. Then we may write

\[
\begin{align*}
&\text{for } x \text{ in } \{\text{for } a \text{ in } \{A\}, \\
&\quad \quad \quad b \text{ in } \{B\} \text{ return encode}(a+b), \text{"tiff"}) \text{ on host_ab} \\
&\quad \quad \quad c \text{ in } \{C\} \text{ return encode}(x + \max(c), \text{"tiff"})
\end{align*}
\]

In this case, the outermost query is executed on the server receiving (or generating it) while the subquery is sent to host_ab which computes the result and sends it back to the parent host.

Also, the WCPS query model is extended with the ability to invoke sub-queries other WCPS services and handle the result of the invocation.

### 4. SAMPLE IMPLEMENTATION

Using the sample query in Section 2 with a function returns the output size of an operator and the assumptions that coverages $A$, $B$, and $C$ are located on hosts host_a, host_b, and host_c respectively, we obtain a query tree furnished with operators output sizes information (Figure 2). Note that the output size of an operator node is recursively computed from the semantics of an operator and input coverages metadata.

![Figure 2: Sample Query Tree Scheduled for Decomposition](image)

---

1 The root node is usually the operator that sends the results to the client
2 A parent sub-graph directly depend on the result of the execution of another sub-graph (which is called the child sub-graph)
our approach is the declarative business process description model [8], where the service designer provides a formal language based process description to services which does the semantic based composition and optimization of the process. Here, the semantics is built in the request model; hence, reasoning is done directly on the code to be executed. The orchestration of our services is based on ActiveXML[1] paradigm, where nested web service calls is embedded in a web service request. However, our nested service calls is embedded in the query as opposed to xml-based embedding in ActiveXML.

5.2 Geoprocessing Services
Different implementation designs have been proposed for distributed geo-processing. [4] and [30] addressed the use of cloud computing for OGC Web Processing Service (WPS)[27] – an OGC standard for a generic service that offers any sort of geo-processing functionality to clients across a network. [13][7][30][16] investigated distributed access to and, processing of geospatial data (using OGC WPS, WCS and WFS standards) within an SOA framework. [3][32], likewise, demonstrated the processing of geospatial data in a grid computing framework. However, these did not sufficiently address the executing dynamically specified processes and workflows. WCPS provides the means of specifying processes declaratively as a WCPS service request (otherwise called WCPS query). Due to the fact that the semantics of the service request is known both to the client and servers, [7] opined that automatic service composition of geo-processing services is possible dispatching, chaining and optimizing is possible. This is comparable to execution model for query processing in distributed databases [25][26][21].

5.3 Graph Partitioning
The WCPS query can be modeled as a graph (see Section 3), hence, our consideration of graph partitioning techniques. This involves division of a graph into a set of approximately equal number nodes such that will minimize a given cost function. Graph partitioning is an NP-hard problem [14][10], and so, all existing partitioning algorithms yield a near optimal solution. A lot of such algorithms exists, examples of these include the Kernighan and Lin algorithm [20], simulated annealing [19], tabu search [29], isoperimetric partitioning [15] etc. However, considering the fact that tree graphs are ordered, the use of these above algorithms is usually unduly expensive for tree graph partitioning. Other algorithms also, exist for assigning each node (clusters of nodes) of a tree graph to an execution host [24][11]. This is suitable for parallelizing operations execution but it has the demerits of large communication and data transport overhead.

6. CONCLUSIONS AND OUTLOOK
Decomposing WCPS services requests into sub-requests and the execution of the sub-requests on different servers in a SOA was studied in this paper. The decomposition aimed at minimizing data transport between the various execution hosts. This is intended to be extended with a more comprehensive cost-based model which makes use of other quality of service functions such as compute and storage capacity, network capacity, etc in the future. This paper also demonstrates that nesting of web service request can be used for the executing dynamically composed
services within a SOA. We, therefore, claim that this approach is flexible, adaptive, and efficient for distributing and executing tasks in a SOA.

7. REFERENCES


