Towards Green Service-oriented Computing

Naseem Ibrahim
Albany State University
Albany, Georgia, USA
naseem.ibrahim@asurams.edu

ABSTRACT

With the wide popularity of distributed and cloud computing a new major concern arise with respect to how green such technologies are. This paper is concerned with providing services over the cloud that are energy-aware. To achieve this goal this paper presents a novel extended service oriented architecture and a novel formal service model. The architecture and the service model extend traditional service oriented architectures to be support context information and mainly location information.

KEYWORDS


1 INTRODUCTION

Green Computing [1] [2] is a computing paradigm with the ultimate goal of eliminating technological waste. In 1992 the U.S. Environmental Protection Agency (EPA) launched a program devoted to the concept of “green” technology, and since the trend of green computing began to evolve from a concept to an architecture. In recent years, the need for green architectures has increased, from issues with energy consumption, landfill waste, toxic waste, and an overall critical matter of resource depletion. Currently, most U.S. computer hardware manufacturers are moving towards producing energy efficient laptops, desktops, and hardware. While this is a step in the right direction, the biggest obstacle lies in the hardware and servers that most corporations use on a day-to-day basis, as well as, how to effectively compute and manage the rate of resource depletion. Throughout the U.S. it is becoming a common practice to be more environmentally aware. Some companies have implemented the practice of using energy efficient hardware, mandating that the hardware be unplugged at the end of a work day, as well as, utilizing the power saving modes on some systems. While these are good ways to conserve energy, there are some companies whose systems are based on data servers and centers.

The most important and vulnerable component in computer system is its data center [3]. “Data center energy and emissions costs are a major concern in green IT analysis because more than half of all IT-related electrical costs are generated there.” [4][5]. A study [4] found that servers coupled with their supporting systems such as cooling units use 100 times the energy per square foot of an office building. IBM [6], a major corporation in Technology reported in 2008 that their, “data centers account for 6 percent of their total floor space but are responsible for 30 percent of their energy cost, and this cost grew at 18 percent.”

This paper is concerned with service provision in the context of cloud computing. In traditional service based computing the matching between service requester’s requirements and available services ignores energy concerns. In other words, the matching is mainly concerned with functionality and price regardless of energy consumption.

The energy consumed by data transfer between service providers and requesters is highly related to the distance between the two parties. The further the distance between the service provider and service requester the higher amount of energy is consumed.

This paper extends traditional service oriented architectures to be more energy-aware. The basic idea is to match services while considering the distance between the service provider and requester. The service requester can set a priority to energy-consumption. Hence, the services that
are closer get a higher chance in being suggested by the matching agency. To achieve this goal this paper introduces new service provision architecture and a new service model.

The rest of this paper is structured as follows. Section 2 provides a brief overview of cloud computing and service-oriented architecture. Section 3 presents the novel extended SOA. Section 4 introduces the novel extended service model. Finally, Section 5 presents some concluding remarks.

2 CLOUD COMPUTING and SOA

While the definition of cloud computing [7] may not be as clear as many would want it, its objectives and characteristics are clear and concise: To provide on-demand access to a network and have on-hand a seemingly infinite amount of resources at the request of the consumer. The cloud computing model has five essential characteristics:

- **On-Demand Self Service**: A user can access resources instantaneously without requiring any human interaction.

- **Resource Pooling**: The resources are grouped together in a centralized location to provide users the resources needed in order to complete the tasks that they need. The resources assigned are determined based on the demands of the consumer. These resources include but are not limited to storage and network bandwidth.

- **Broad Network Access**: The computing resources that meet the demands of the consumer are delivered to them through a network, such as the internet, and are access through heterogeneous platforms (e.g., mobile phones, laptops, tablets).

- **Rapid Elasticity**: Computing resources to the consumer appear to be automatically present when they need it. The consumer is allowed to use resources in order to increase or reduce their scalability whenever they please based on their demand. This is mostly because the provisioning of the resources to them appears to be infinite so the consumer can appropriate the resources in any quantity at any time.

- **Measured Service**: The Cloud system has the capability to measure the usage of the resources in order to control and optimize the use of it by the consumer. Even though there are multiple consumers that end up using the resources in the cloud, the measuring capability is able to gauge the level of use with the type of the service (e.g., network bandwidth, software us, processing).

Along with the above five characteristics of cloud computing comes three service models, these service models can be summarized in [8]:

- **Software as a Service (SaaS)**: The consumer is able to release their application on a hosting environment that can be used by the clients through networks, such as a web browser or PDA. The consumer has no control over the infrastructure of the cloud since it usually employs multi-tenancy (multiple client-organizations at once) architecture.

- **Platform as a Service (PaaS)**: The consumer deploys applications onto the cloud infrastructure. These applications, unlike the SaaS, do not have to be completed applications but can be in-progress applications that are created by the consumer using programming languages and tools supported by the cloud’s provider.

- **Infrastructure as a Service (IaaS)**: The consumer is able to deploy and run arbitrary software onto the software. The consumer is able to control the operating system and to the network settings, but still cannot do anything to the infrastructure of the cloud.

In this paper we focus on SaaS. The main realization of SaaS is Service-oriented Computing (SOC). SOC [9] is a computing paradigm that uses service as the fundamental element for application development processes. The primary goal of SOC is a rapid, low-cost development of distributed...
service applications in heterogeneous environments. An architectural model of SOC in which service is a first class element is called Service-Oriented Architecture (SOA) [10]. In a SOA it should be possible to define, update, compose, communicate, and deliver a multitude of services. Many SOAs for an application are possible, however they may differ in the way they define services, define service operations, and realize them in a practical setting. Yet, all SOAs must agree on the following three distinctive characteristics:

- services are the basic constructional elements,
- service interactions are standard-based, and
- SOA is both dynamic and evolving.

These three characteristics distinguish SOA from traditional software architecture [11]. Consequently, SOA demands a new approach to conceptualize services in order to weave them together to meet an application.

According to [10], a service contract establishes the terms of engagement with the service, provides technical constraints and requirements, and any semantic information the service provider wishes to make public. The essential structural aspects of a service contract [13] are interface, functionality, protocol, and quality of service (nonfunctional) requirements.

- **Interface**: It defines the syntactic communication abstraction for service request and service response.
- **Functionality**: It precisely states what a service can do for a user. It is the set of operations provided by a service. Each operation can be specified using preconditions and postconditions. All preconditions must be true when a service operation is called. All postconditions must be guaranteed to be true after the service is successfully invoked.
- **Protocol**: It is the behavior of the service in terms of the input messages (requests) and the output messages (responses).
- **Nonfunctional properties**: Quality of service features are the nonfunctional properties of the service. In principle, these include performance, reliability, availability, security, and accessibility.
The question we ask “can the current SOA paradigm be used, as is, to provide services that are energy-aware?” We believe the answer to this question is NO.

The current structure of service contract is inadequate to fully specify location information. Hence, in this paper we extend the definition of a service with the context.

Current service models can be classified based either on the language, the architecture or a combination of both used to describe service. The two main languages that have been used for modeling services are UML [14], [15], and WSDL with the related Web description languages [16], [17], [18], [19]. Architecture based service modeling approach uses an Architectural Definition Language (ADL) [11], [20] to describe services. There are a few methods [21], [22] which use both language and some abstract architectural details for describing service features. All these approaches model the functional behavior of services. Nonfunctional properties are only supported in a simple manner by few approaches. Contextual information is not represented by any approach. Hence, in the rest of this paper we propose a new extended SOA and a new extended service model.

3 ENERGY-AWARE SOA

In SOA, the service requester queries for available services that meet its requirements. Such services are provided by multiple service providers on multiple locations. A Service broker (service registry) is usually responsible for matching service requester needs with available services. In this paper we propose the consideration of data consumption as a main factor in the matching process. The service provider when publishing the service will specify the location of the service. The service requester will also specify the location of service provision. Hence, the service registry will be able to match the service requester requirements from the list of available service while reducing energy consumption. This is done by reducing the distance the data travels between the service provider and the service requester. The less distance a data travels the less energy is consumed. The rest of this section introduces the extended energy-aware service-oriented provision architecture (EASOA). EASOA will enable service providers to define energy-aware rich services for the provision of energy-aware services. It will also enable service requesters to obtain services that best models their requirements while considering their energy consumption concerns.

Figure 2 shows the architecture of EASOA. EASOA consists of the following elements:

- **Service Requester**: It is the entity that is requiring a service. It represents the client side of the interaction. It can be an application or another service.
- **Service Provider**: It is the entity that provides an implementation of a service specification. Service providers publishes service descriptions as ExtendedServices on registries to enable automated discovery and invocation.
- **Energy-aware requirement definition**: In the new extended SOA model, a service requester is able to specifically define and list all energy requirements and constraints in a rich definition. These requirements and constraints are defined and stated as a requester context.
- **Energy-aware service definition (ExtendedService)**: traditional SOA service definitions, such as WSDL, focus on service functionalities and some nonfunctional properties. Our definition of service is much richer. The service
definition contains enough information to facilitate a better matching between service requester requirements and the service provider services. This is achieved by introducing context as a first class element in a service definition. This context information is energy-aware. It lists the different learner requirements that it can meet.

- **Context-aware service registry (CASR):** Traditional service registries publish services focusing on service functionalities. In CASR, the published service is much richer. It takes into consideration the contextual information of the service requester and provider.

- **Energy-aware service matching:** This unit is responsible of two main roles. First, it matches the requirements of the service requester to the available services in CASR. The novelty in the matching process is that it does not only focus on the functional requirements but it rather takes into consideration the context information of the service requester. Second, it adapts to the changing requirements of the service requester during service execution.

4 EXTENDED SERVICE

One of the main elements of the EASOA is the Energy-aware service definition. In this section we present an informal and a formal based definition of the extended service model.

4.1 Informal Definition of ExtendedService

An ExtendedService is divided into the two main parts Service and Contract as shown in Figure 3.

The **Service** section has the three parts Functionality, Nonfunctional properties and Attributes.

![Figure 3. ExtendedService Architecture](image)

1. **Functionality:** Its definition includes the function signature, result, precondition and postcondition. The signature part defines the function identifier, the invocation address, and the parameters of the function. The function invocation has the same effect as in a programming environment, since service function is an autonomous program. Each parameter has an identifier and a type. The result part defines the returned data of the service function. The precondition should be made true, either by the service provider or the consumer, in order to make the function available. The postcondition is guaranteed by the service provider to be true after service execution.

2. **Nonfunctional properties:** The nonfunctional properties associated with the service are listed in this section. Pricing information, which can itself be a complex property expressing different prices for different amount of buying, is an example of nonfunctional property. For some types of services, the amount of storage required and speed of downloading may be included as nonfunctional properties.

3. **Attributes:** Every attribute is a type-value pair. Attributes provide sufficient information that is unique to a service. As an example, for providing a course the
appropriate attributes may include title of course and institution name.

The **Contract** is divided into the two main parts Legal Issues and Context.

1. **Legal issues**: Business rules and trade laws that are enforced at the locations of service provision and service delivery are included in this section. Example policies govern refund, administrative charges, penalties, and service requester’s rights. Such rules are expressible as logical expressions in predicate logic.

2. **Context**: The context part of the contract is divided into context info and context rules. The contextual information of the service provider is specified in the context info section. The situation or context rule that should be true for service delivery is specified in context rules section. It is the responsibility of the service requester to validate the context info for obtaining the service, and it is the responsibility of the service provider to validate the context rules at service delivery time.

4.2 Formal Definition of ExtendedService

An ExtendedService can be formally written using a model-based specification notation. For example, the service functionality can be written in predicate logic, the data section can be formalized as an aggregated abstract data type, the nonfunctional properties, and legal rules can be written as logical expressions. Thus, the contract part is a collection of logical formulas. The context information is written in the notation introduced by Wan [2]. Because of this underlying formalism it is possible to rigorously verify the claims made in an ExtendedService. Below we briefly outline the formal context notation and we give the formal representation of the rest of ExtendedService elements.

**Context Formalism**

Context information is formally specified, as defined in [2], using dimensions and tags along the dimensions. In our research, we have been using the five dimensions WHERE, WHEN, WHAT, WHO, and WHY. The dimension names are generic. For example, the dimension WHERE is associated with locality, the dimension WHEN is associated with temporal information such as time and date, the dimension WHO is associated with subjects (or roles), the dimension WHAT is associated with an activity, and the dimension WHY is associated with a purpose for the stated activity. In general, it is the responsibility of service providers to choose as many dimensions and their names in order to present the contexts associated with services.

Assume that the service provider has invented a finite set $\text{DIM} = \{X_1, X_2, \ldots, X_n\}$ of dimensions, and associated with each dimension $X_j$ a type $\tau_j$. Following the formal aspects of context developed by Wan [2], we define a context $c$ as an aggregation of ordered pairs $(X_j, v_j)$, where $X_j \in \text{DIM}$, and $v_j \in \tau_j$.

A context rule is a situation which might be true in some contexts and false in some others. For example, the situation \(\text{VERYWARM} = \text{Temp} > 40 \land \text{Humid} > 70\), is true only in contexts where the temperature is greater than 40 degrees, and the humidity is greater than 70.

**A Model-based Formalization of other ExtendedService Elements**

After explaining the basic notation we build the model incrementally, according to the template in Figure 3. A constraint is a predicate logic expression, defined over data parameters and attributes.

Let $C$ denote the set of all such logical expressions. $X \in C$ is a constraint. The following notation is used in our definition:

- $T$ denotes the set of all data types, including abstract data types.
- $D_t \in T$ means $D_t$ is a datatype.
- $v : D_t$ denotes that $v$ is either constant or variable of type $D_t$.
- $X_v$ is a constraint on $v$. If $v$ is a constant then $X_v$ is true.
- $V_q$ denotes the set of values of data type $q$. 

ISBN: 978-0-9891305-3-0 ©2013 SDIWC
• x :: Δ denotes a logical expression x ∈ C defined over the set of parameters. A parameter is a 3-tuple, defining a data type, a variable of that type, and a constraint on the values assumed by the variable. We denote the set of data parameters as Λ = {λ = (Dt, v, Xv) | Dt ∈ T, v : Dt, Xv ∈ C}.

Service Definition

1. **Functionality:** An ExtendedService provides a single function. This functionality is defined to include the function signature, result value, preconditions and postconditions.

   **Definition 1:** A service function is a 4-tuple f =<g, I, pr, po>, where g is the function signature, i is the function result, pr is the precondition, and po is the postcondition. A signature is a 3-tuple g =<n, d, u>, where n: string is the function identification name, d = {x|x ∈ Λ} is the set of function parameters and u: string is the function address, the physical address on a network that can be used to call a function. For example, it can be an IP address. The result is defined as i = <m, q>, where m:string is the result identification name and q = {x|x ∈ Λ} is the set of parameters resulting from executing the ExtendedService. The precondition pr and post condition po are data constraints. That is, pr :: z, z ⊆ Λ and po :: z, z ⊆ Λ.

2. **Nonfunctional properties:** Typical nonfunctional properties associated with the service are pricing and maintenance information. Pricing can be formalized as follows.

   **Definition 2:** Nonfunctional property list is κ =<p, ...>, where p is the service cost and ... denote other nonfunctional properties. The service cost p is defined as a 3-tuple p = <a, cu, un>, where a : N is the price amount defined as a natural number, cu : cType is currency tied to a currency type cType, and un : uType is the unit for which pricing is valid. As an example, p = (100; $; hour) denotes the pricing of 100$/hour. Other nonfunctional properties can be similarly defined using appropriate data types and included in κ.

3. **Attributes:** These include some semantic information that is unique to a service.

   **Definition 3:** An attribute has a name and type, and is used to define some semantic information associated with the service. As an example, each ExtendedService can be given a unique identifier, a version number, and type of release. They are defined as service attributes. The set of attributes is α = {Dv, vα | Dt ∈ T, vα : Dt}.

Putting these three definitions together we arrive at the formal definition of a service given below.

**Definition 4:** A service is a 3-tuple σ = <f, κ, α>, where f is the service function, κ is the set of nonfunctional properties, and α is the set of service attributes.

Contract Definition

1. **Legal issues:** As part of the contract in an ExtendedService, a set of legal rules that constrain the contract may be included.

   **Definition 5:** A legal issue is a rule, expressed as a logical expression in C. A rule may imply another, however no two rules can conflict. We write l = {y|y ∈ C} to represent the set of legal rules.

2. **Context:** Both context information and context rules are formally specified in a contract. These two parts provide context-awareness ability to ExtendedService.

   **Definition 6:** A context is formalized as a 2-tuple β =<r, c>, where r ∈ C, built over the contextual information c. Context information is formalized using the notation in [2]: Let τ : DIM → I, where DIM = {X1, X2,...,Xn} is a finite set of dimensions and I = {a1, a2,...,an} is a set of types. The function τ associates a dimension to a type. Let τ(Xi) = ai, ai ∈ I. We write c as an aggregation of ordered pairs (Xj, vj), where Xj ∈ DIM, and vj ∈ (Xj).

   Formally a contract is defined below.

   **Definition 7:** A contract is a 2-tuple μ =<l, β>, where the set of legal rules l and β the context are defined as presented above.

Putting these definitions together we arrive at a formal definition for ExtendedService.

**Definition 8:** A ExtendedService is a 2-tuple s =<μ, σ>, where μ is a contract, and σ is a service.
5 CONCLUSIONS

This paper introduced a new novel extend service oriented architecture for the provision of energy-aware service. To support the novel architecture it provided a formal service model that supports the specification of context information and mainly location information. Our ongoing work includes a complete implementation of the proposed architecture.

6 REFERENCES


