In this homework assignment, you must choose one of the three case studies that are provided as appendices to this assignment: Big Data, C4, or BOINC. Each case study describes a scenario pertaining to a large-scale software system.

Your assignment will be slightly different depending on your choice case study, but all three options are comparable in difficulty. These case studies aim to help you understand how to do the following:

- Decompose the description of a desired system and its requirements into components and connectors.
- Understand how designs you come up with typically exhibit one or more architectural styles.
- Weight the tradeoffs of selecting one architectural style over another.
- Understand how your design succeeds or fails in satisfying requirements and/or addressing architectural challenges.
- Explain how a use-case scenario relates to the architecture that you design.

Your choice of a particular case study will in no way impact your grade on this assignment. Solving more than one assignment is not expected and will result in no extra credit. Please focus on a single case study.

You must clearly indicate which case study you have selected.

The last question of every case study has a corresponding supplementary appendix titled *Architecture and Security*. Please consult this appendix before answering the question.
Big Data Case Study Assignment

In this case study, we will focus on selecting and designing a set of components, connectors, and architectural styles that satisfy the requirements and complete the challenges of the “Big Data” Bioinformatics System, hereafter referred to as the BD system.

1. Draw a diagram of your BD system architecture. Include the components, connectors, and associations between components and connectors in your diagram. Please ensure that your diagram is readable. Note that there is no such thing as the “correct” or “optimal” architecture. However, as a granularity guideline, your decomposition of BD should consist of no less than 15 distinct components.\(^1\) **Submit one diagram.**

2. Provide a description of each component and connector. Your description should name each component, explain the key functionalities of each component, and explain the set of connectors selected for the system. **We recommend that you answer this question as a bulleted or numbered list or in the form of a table.**

3. Two architectural styles (e.g., layered or piper and filter) should be selected as part of your design. Describe the style constraints (e.g. types of components and connectors and constraints on which of these elements can communicate with each other) and the non-functional properties exhibited by the style. Explain which set of components and connectors from your design adhere to the style constraints of your selected styles. If you have decided to include a connector not typically associated with your selected architectural styles, please explain why you chose to deviate from your chosen styles. **Please limit your answer to 2 paragraphs.**

4. For each of the two styles you selected, select a different style and describe at least two advantages your selected style has over this new style. In total, you will be considering four styles and describing four advantages. **Please limit your answer to 1 paragraph.**

5. Select a subset of your components and connectors and describe how they satisfy two of the requirements of the BD system. Please avoid saying that all your components and connectors together satisfy the two requirements. Instead, describe how the functionalities or properties of each component and/or connector work together to satisfy a particular requirement. **Please limit your answer to 1 paragraph.**

6. Select a subset of your components and connectors and describe how they deal with the two of the key architectural challenges of the BD system. Please avoid saying that all your components and connectors together deal with the two architectural challenges. Instead, describe how the functionalities or properties of each component and/or connector work together to complete a particular challenge. **Please limit your answer to 2 paragraphs.**

---

\(^1\) This is only a guideline. There is nothing magical about this number, nor do we have a specific preferred solution in mind.
7. Come up with a use-case scenario for the BD system then describe each component and connector involved in that use case scenario. If you can only think of a use case scenario that involves only two components, then describe a second use case scenario involving another two components. Please limit your answer to 1 or 2 paragraphs.

8. Select two types of vulnerabilities from the list of vulnerability types (CWEs) in *Encrypt Data*\(^2\) that you think may potentially occur in your system. Do your proposed architectures protect from those vulnerability types? Explain why or why not. Please limit your answer to 1 paragraph.

---

\(^2\) [https://cwe.mitre.org/data/definitions/1013.html](https://cwe.mitre.org/data/definitions/1013.html)
C4 Case Study Assignment

The Call Center Customer Care (C4) Case Study provided as an appendix to this assignment presents an initial high level (“Level 1”) architectural breakdown for the system used by a large telecommunications company. This system comprises several subsystems, one of which is C4 itself.

1. The lectures and readings have begun discussing the architectural design process, including the selection of architectural styles. Not all styles are applicable to all systems and any choice of style will involve making trades on various system attributes. Pick any two styles that you have read about and design two different architectures for the C4 system, one that adheres to each style. In order to apply a style, you will need to create a detailed architectural breakdown (a “Level 2” architectural breakdown) for C4. In other words, “expand” the C4 box shown the figure present on page 1 in the Case Study into an architecture. It is, of course, difficult to decide on the exact degree of detail to be provided in a “Level 2” architecture, such as the one required for this assignment. Make sure to show all the connectors inside the C4 architecture as well as those that interconnect C4 to the other parts of the system. Moreover, you should graphically distinguish different types of the connectors used in your design. Also, there is no such thing as the “correct” or “optimal” architecture. However, as a granularity guideline, your decomposition of C4 should consist of no less than 15 distinct components. You are not required to select a style from the course text book, but you must let us know what style you are attempting to apply and provide a reference if the style is not one from the course text. Submit one diagram for each architecture you design.

2. Give a brief rationale for your architecture: why did you select the styles that you did? Weigh the pros and cons of each architectural style. We will not grade you based on how accurately you apply each style so much as your rationale for selecting a particular style and understanding its limitations. Please limit your answer to 2 paragraphs.

3. Compare each of your architectures: give one example of a system property/requirement described in the Case Study that is addressed in a superior manner by one of your architectures. Be sure to not only name the property/requirement, but explain how each architecture addresses the property/requirement and give rationale for why you think one system is superior to the other. Please limit your answer to 2 paragraphs.

4. Since C4 is a very large system with many different, possibly conflicting, requirements, your architecture may only directly address a subset. To demonstrate this, for one of your architectures, select two of the key architectural challenges and requirements (listed in bulleted items on page 4 of the Case Study) and argue/discuss how your architecture DOES NOT support them in an acceptable fashion. Please limit your answer to 1 paragraph.

---

3 This is only a guideline. There is nothing magical about this number, nor do we have a specific preferred solution in mind.
5. Modify one of your architectures to address the shortcomings discussed in the preceding question. Can you do so without violating the chosen architectural style? Give the rationale as to whether you think this change is appropriate to make for the system. Please limit your answer to 1 paragraph and a new diagram.

6. Add a capability to both your architectures to log interactions between C4 and Downstream Systems. Logging should be enabled only when there are less than 100 active users interacting with the system concurrently. How will you change each architecture in response to this new requirement? Which architecture can be changed more easily? Why? Please limit your answer to 2 paragraphs and, if you find it necessary, one diagram for each of your architectures.

7. Select four types of vulnerabilities, two from the list of vulnerability types (CWEs) in Authenticate Actors\(^4\) and two from the vulnerabilities in Validate Inputs\(^5\) that you think may potentially occur in your system. Do your proposed architectures protect from those vulnerability types? Explain why or why not. Please limit your answer to 2 paragraphs.

---

\(^4\) [https://cwe.mitre.org/data/definitions/1010.html](https://cwe.mitre.org/data/definitions/1010.html)

\(^5\) [https://cwe.mitre.org/data/definitions/1019.html](https://cwe.mitre.org/data/definitions/1019.html)
BOINC Case Study Assignment

The Berkeley Open Infrastructure for Network Computing (BOINC) Case Study, provided as an appendix to this assignment, presents an initial high-level (“Level 1”) architectural breakdown for the system. The overall BOINC system comprises several components, one of which is the BOINC task server.

1. The lectures and readings have begun discussing the architectural design process, including the selection of architectural styles. Not all styles are applicable to all systems and any choice of style will involve making trades on various system attributes. Pick any two styles that you have read about and design two different architectures for the BOINC task server, one that adheres to each style. In order to apply a style, you will need to create a detailed architectural breakdown (a “Level 2” architectural breakdown) for the task server. In other words, “expand” the task server box shown in Figure 4 in the Case Study into an architecture. Make sure to show all the connectors inside the task server architecture as well as those that interconnect the task server to the other parts of the system. Moreover, you should graphically distinguish different types of the connectors used in your design. Also, there is no such thing as the “correct” or “optimal” architecture. However, as a granularity guideline, your decomposition of the task server should consist of no less than 15 distinct components. You are not required to select a style from the course textbook, but you must let us know what style you are attempting to apply and provide a reference if the style is not one from the course text. Submit one diagram for each architecture you design.

2. Give a brief rationale for your architecture: why did you select the styles that you did? Weigh the pros and cons of each architectural style. We will not grade you based on how accurately you apply each style as much as your rationale for selecting a particular style and understanding its limitations. Please limit your answer to 2 paragraphs.

3. Compare each of your architectures: give one example of a system property/requirement described in the Case Study that is addressed in a superior manner by one of your architectures. Be sure not only to name the property/requirement, but also to explain how each architecture addresses the property/requirement and give rationale for why you think one architecture is superior to the other. Please limit your answer to 2 paragraphs.

4. Since BOINC is a very large system with many different, possibly conflicting, requirements, your architecture may only directly address a subset of those requirements. To demonstrate this, for one of your architectures, select two of the key architectural challenges and requirements (listed in bulleted items in the last two pages of the Case Study) and argue/discuss how your architecture DOES NOT support them in an acceptable fashion. Please limit your answer to 1 paragraph.

---

6 This is only a guideline. There is nothing magical about this number, nor do we have a specific preferred solution in mind.
5. Add a capability to both your architectures to integrate one or more cloud(s) into the
BOINC computation. The project operator must be able to vary the amount of cloud
computation power involved in the BOINC computation based on the active volunteer-
participant numbers. How will you change each architecture in response to this new
requirement? Which architecture can be changed more easily? Why? Please limit your
answer to 2 paragraphs and, if you find it necessary, one diagram for each of your
architectures.

6. Select four types of vulnerabilities, two from the list of vulnerability types (CWEs) in
Limit Access\(^7\) and two from the vulnerabilities in Validate Inputs\(^8\) that you think may
potentially occur in your system. Do your proposed architectures protect from those
vulnerability types? Explain why or why not. Please limit your answer to 2
paragraphs.

\(^7\) https://cwe.mitre.org/data/definitions/1015.html
\(^8\) https://cwe.mitre.org/data/definitions/1019.html
APPENDIX 1: Architecture and Security

Software Vulnerabilities

Software vulnerabilities, defined as errors in the code that create potential for attacks, cause significant damage if not found and fixed before malicious actors come across them. Fighting vulnerabilities has become so important that the U.S. government sponsors a database, the National Vulnerability Database (NVD), where information about vulnerabilities is collected and shared, in hopes to educate the broader community. In NVD, each manifestation of a vulnerability is given a unique ID, referred to as the Common Vulnerabilities and Exposures (CVE) ID. Moreover, the vulnerabilities are categorized further by broader categories from the Common Weakness Enumeration list, a community based categorization of vulnerabilities. The CWE list has three broad groups of vulnerabilities: research concepts, development concepts, and architectural concepts. Examples of specific vulnerability types include Improper Authentication (CWE-287), Cross-Site Request Forgery (CWE-352), and Improper Restriction of Operations within the Bounds of a Memory Buffer (CWE-119).

For example, information about one of the vulnerabilities of Tomcat, the Apache Web Server, can be found here. In this page, you can see the vulnerability’s CVE, its categorization or CWE, and other relevant information.

Research into how vulnerabilities come to be has shown that a significant number of vulnerabilities arise due to architectural decisions: Either an architectural decision is missing, is incorrect, or is improperly implemented in the code. Compared to vulnerabilities that happen due to code errors, the ones that happen due to architectural decisions are more subtle and sophisticated, making them more challenging to fix. As such, the community is pushing for “secure-by-design” products, where preventing these vulnerabilities becomes the focus of architects and developers from the early stages of the development life-cycle.

Example of an Architectural Vulnerability

When creating the architecture of a client/server product, an important architectural consideration is user authentication and where it is going to happen. A common architecture weakness is to locate user authentication in the client code, but not the server code. This is referred to as the Use of Client-Side Authentication vulnerability type. This is a vulnerability because the client can be modified to bypass the authentication check completely by someone with knowledge in reverse-engineering.

For more information, refer to:


doi: 10.1109/ICSAW.2017.25
APPENDIX 2: The “Big Data” Bioinformatics System

Introduction

The purpose of this document is to describe a fictitious bioinformatics research system based in part on the design and implementation of a similar system for a local Hospital. The document is divided into five parts. The initial section describes the motivation for the bioinformatics system, called “Big Data” Bioinformatics System or “BD” for short. A set of motivating requirements follows. After the set of motivating requirements, a set of proposed architectural challenges is presented. The key interactions for the system components are presented second to last, and the final section describes the intended use of the Apache OODT software system in order to meet the needs and challenges of the proposed BD system.

Motivation

We are tasked with helping researchers at a local hospital analyze information collected (with consent!) from patients in the critical care unit. The various forms of information are shown in the left portion of Figure 1 below.
Information collected includes:

1. Long-term **Historical Patient Records** including admit date, and discharge date, disease type, drugs administered and other discrete variables for a set of 10,000 patients who have visited the hospital in the last 10 years.

2. Real-time, patient **Monitor data** from various instruments by the patient beside. These instruments include heart monitors, oxygen level monitors, monitors for lung function and brain activity and other instruments. This particular set of information about a patient is only collected short-term, over the past 2 years, because only some of the newer instruments have the ability to digitally record and store this data. This information is relevant for about 2000 patients who have stayed at the hospital.

3. Long-term, **Doctor’s and Nurse’s notes** recorded during a patient’s visit. This mostly includes free-text notes electronically captured by machines near the patient beside, where the nurses and doctors record their observations about a patient. The dataset is unstructured and may contain overlapping information with the **Historical Patient Records** and **Monitor data** information. This information covers around 10,000 patients over 10 years.

4. Short-term, **Offline Research** databases, containing information collected during clinical trials and destroyed after a period of one year. This information is provided for about 300-500 patients in the hospital.

These rich datasets collected about patients are used by researchers and clinicians within the hospital to perform scientific research with the goal of identifying causalities and observations regarding patient mortality: in other words, this information is used to help identify and observe characteristics about patients who lived and died. These characteristics can feed into further research that hopefully one day can allow doctors to make informed decisions based on observations (about patient’s heart rate, about their appearance, etc.) that lead to better overall hospital care.

The immediate steps required to bring forth this proposed future more immediately involve the construction of a *decision support biomedical informatics system* which we have dubbed “Big Data” Bioinformatics System or “BD” for short. BD helps hospital researchers collect the information as different types of files (labeled as *f* in the middle portion of Figure 1), which are then combined and processed and eventually shoved into “research databases” on the middle-right portion of Figure 1. Research databases are focused data warehouses containing combined information from the above four data sources that can easily be used by hospital researchers for a particular scientific study (resulting in some publication), or in generating raw data that can be shared with other hospitals and the broader community, or in pulling into custom analysis environments like IDL, Matlab or R for further analysis.

**Requirements**

BD has several over-arching requirements as outlined by the hospital administrators and other stakeholders in the system.

1. *The patient information must be kept secure* – if there were an “unbreakable” requirement, this would be it. Because of U.S. HIPAA law and despite the patients signing consent forms, all of the information, whether at the data source (left hand side of Figure 1), whether in file form (middle portion of Figure 1), whether in the research databases (middle-right portion of Figure 1), or while in transit to the broader community or to the analysis tools, the data must be protected using encryption, using secure-by-design principles, and using best practices.

2. *The original data sources cannot be modified, only read* – information cannot be updated in any one of the four data sources on the left hand side of Figure 1. The information can only be read. The same goes for the Big Data file system portion – research data files can only be read, and those files should only be created once, with no append capability after their initial creation.
3. *Research databases may be dynamic* – Research databases are needed only for particular research campaigns and case studies, and may be stood up for short-term (monthly), or longer-term (multi-month, multi-year) purposes. All research databases will eventually be destroyed in order to comply with security requirements.

4. *Data must be easily accessible from research databases* – Research database information is fed into scientific papers, is fed into scientific tools, and is distributed to other hospitals for multi-center collaborations. It must be easily accessible, in standard formats, and secure. Desired tool support includes: R, Matlab, and IDL, as well as common CSV formats.

5. *Data must be de-identified* – In keeping with the security requirement, all original patient identifiers must be de-identified, and obfuscated from traceability back to their patient origin. This is strictly true for all data in research databases, but may not be true in the Big Data file system.

6. *A standard data model and metadata model should be used to describe patient information* – The hospital has been looking at SNOWMED, HL7 and other models and formats, but does not strictly require use of any of those. They are simply provided as examples.

7. *Data storage may require large scalability bursts, depending on upcoming research conferences, and depending on hospital priorities* – there may be on the order of 50-100 research databases constructed within a month’s time depending on the importance of an upcoming research conference, or collaboration with another hospital.

8. *Processing scalability will be significantly less than data storage* – Though the researchers want to run data combination algorithms, and identify things like missing “data” (as we will elaborate below), this processing will be significantly more long-term and less “bursty” than the data storage. The important thing to the hospital researchers is the availability of the raw information as re-processing is always an option.

**Key Architectural Challenges**

Given the above requirements, there are several key architectural challenges with the design of the proposed BD system. We will elucidate those challenges below:

1. *Creation of a read-only large-scale distributed “copy” of the origin data.* The researchers do not have direct access to the Historical Patient Records, the Monitor data or any of the other two data sources available for research and analysis. Each curator of information has expressed her concern to the hospital director of research mainly amounting to growing concern of real-time queries to their data sources. Each one of the data sources has taken years to build up, and the technology that captures the information was never built to respond to the needs of the hospital research community inasmuch as the data sources were constructed to provide improved and immediate benefits to patient care. Because of this, the data providers (on the left side of Figure 1) have agreed to have their information periodically “dumped” into the Big Data filesystem, under the control of the hospital researchers, where they can experiment and analyze the information to their heart’s delight.

2. *Keeping the BD Filesystem in Sync and “Fresh”* – This isn’t exactly straight-forward, and it’s really dictated by the types of analyses required, and presence of “missing data” which may periodically occur when performing the data export from any of the available data sources. Monitor data is sketchier since it’s kept in a cache whereas data such as the Historical data are stored in more reliable database systems like Oracle or MySQL.

3. *Each original data source uses a different technology* – as can be gleaned from the discussion thus far, each of the original data sources uses a different technology – some like the monitoring data use a proprietary vendor cache technology, unique to each instrument. Others, on the other hand, use commercial DBMSs like Oracle (Historical data). Others use MySQL. Yet others are stored in note taking systems, sometimes called EHR or electronic health care records systems. BD needs a unified way of dumping information into easily accessible files.
4. *The data models look very different for each type of file* – The BD filesystem houses all of the aforementioned data sources, and each one of the output files from them looks vastly different. Monitor records look similar to CSV output, with blood pressure, heart rate, and other variables in comma separated lists. The data from the doctor and nurse’s notes system looks like free-text output; the data from the historical patient records table looks like database row output. Each one of these files must be combined into the specialized research databases.

5. *Making the data available for scientific, peer-reviewed research and dissemination to other hospitals requires explicit permissions and necessary security in place at each component in the system*. None of the components can escape it – they all must be secure.

6. *The analysis tools accept data in different ways* – R, Matlab, and IDL all read data differently and specialized output handlers for the research databases must be made in order to export to these tools.

7. *Algorithms must be written to combine the data in unique ways, fill in missing data, and to de-identify the data*. The algorithms need to be scheduled and run efficiently too, especially during paper writing campaigns of the researchers where resources are in contention.

**Key Interactions**

Of course, with the existing description of the system, the complexities of the components and the data flow and the interactions can probably be discerned as significant. To start, the BD file system will need to pull data from the origin 4 data sources, after which it needs to push data to allow the construction of the research databases. Research databases are likely built from some pre-defined workflow involving data de-identification, combination and summarization. After the research databases are created, their information is made available via client pull (from an interested scientist or researcher into e.g., Matlab or IDL or R), or via push to an external hospital on an external disk or via electronic transfer for further analysis. The research database information may also be made available to use as raw data in a (set of) scientific research paper(s).

**Apache OODT**

The hospital researchers have selected Apache OODT as the implementation technology to construct the BD system. OODT provides a set of components that allow data integration, information retrieval and extraction, and data dissemination. OODT is known as a “grid” technology as it allows the constructions of virtual organizations of users sharing data, computation and resources alike.

The BD system should be built using Apache OODT components. The Apache OODT website is located at: [http://oodt.apache.org](http://oodt.apache.org).

A full treatment of Apache OODT is beyond the scope of this description document. We point the interested readers to the following 2 papers:


APPENDIX 3: Call Center Customer Care System
(a case study)

Note: this is a simplified and generalized description of a real system

Introduction

The Call Center Customer Care System has been developed by Andersen Consulting for a large US telecommunication company. The primary function of the system is to support interactions with the customers that request new services (ex: new phone lines), changes in the configuration of the existing services (ex: phone number changes, long-distance company changes, or relocation), or report problems.

The phone company has over 19Mil customers. Considering how often, on average, a customer changes his/her service configuration, the system has to support up to 400 company representatives simultaneously at near 7X24 availability level. However, these representatives are not the only means by which a customer can request a change or report problems. For example, there exists a phone service (Quick Service) by which customers can communicate with the system. This is discussed in more detail later.

System Interactions

The C4 (Call Center Customer Care System) system interacts with a number of other systems, in particular with:

- A network provisioning system that makes physical changes to the network configurations and supports network management
- A Billing system
- A host of corporate DBs, and
- A number of downstream systems.

A high level structure of the system interactions is show in Fig. 1. Collectively, this is a big thing and rather complicated.
Interactions with NOSS

NOSS (Network Operations Support System) is the network management and provisioning system. It is being developed concurrently by a large 3rd party hardware/software company specializing in communication networks. Its functionality contains:

- Work force management - management of maintenance crews
- Provisioning - putting physical network components in place such as connections from the curb to the house
- Network creation - an peculiar name for maintaining information about new and existing physical network
- Activation - automatic turning on/off of services
- Network management that contains:
  - status monitoring and measurements
  - proactive maintenance
  - diagnostics, and
  - problem reporting, and
- Field access - a subsystem providing up-to-the-minute information for field technicians.

The NOSS architecture is based on the OSI CMIS (Common Management Information Service). The interface between C4 and NOSS is a large set of messages in the named-tag/value format. Each message inter-change is technically a synchronous pair of messages composed of a request and a reply. At the application level there are two types of interactions: 1) a request followed by a response that contains the requested information and/or confirmation of NOSS action, and 2) an overall interaction consisting of two message pairs; a) initial request followed by a reply containing a request identifier, b) an unsolicited message from NOSS containing the previously issued request identifier along with related data followed by a C4 reply acknowledging receipt of message.

The basic interactions between C4 and NOSS are as follows:

- C4 sends a Service Request (service order messages) to NOSS to perform service reconfigurations
- C4 send queries to NOSS about existing network status or capabilities. This is usually done while an agent is talking to a customer. For example, a request may be sent to NOSS about availability of service at a particular geographic address/service location or possible service activation dates, and
- C4 may request from NOSS to “lock” resources such as phone numbers for a service request.

Interactions with Downstream Systems

Downstream Systems are systems such as: Long-distance Carrier Services, 911 Service, Voice Mail, Interfaces to Phone Directory, and Revenue Collections (credit scoring and checking).

The C4 interacts with these systems in tow ways:

- It publishes requests via the Publish/Subscribe (P/S in Figure 1) component that the downstream systems should react to. For example, all new phone connections are published so the 911 service is connected to them in the required number of hours
- The downstream systems notify C4, via a direct asynchronous message, about significant business event that effect customer configuration. For example, the Long-distance Provider system may notify C4 that a client should be connected to a new long-distance company.

Note: one of the design challenges that we will discuss later was that direct requests from a customer may conflict with a similar request coming from an external system. For example, a customer may call to request a change of their long-distance carrier, say from Sprint to MCI, at the same time ATT sends notification that the same customer has now selected them as their long-distance carrier.

Interactions with corporate DBs and Billing
Some of the major functions of the billing system are: (1) invoice calculation for local services, (2) invoice printing for both local and long-distance services, (3) revenue reporting, and (4) bill inquiry and adjustment.

C4 does not interact with the Billing System, but it works against the same set of DBs. More specifically, C4 interacts with the on-line part of the corporate DBs, while the Billing Systems work with the batch copy of the on-line DBs (this is shown in Figure 1). Monthly billing is done in a number of cycles. Each cycle processes billing information of a set of customers. All updates to the data of the current cycle are halted (applied only to the on-line data) until the end of the cycle. The updates (at the end of the cycle) are done in batch and are transparent C4.

All customer data are stored in over 100 tables. C4 does not use all of them. During any customer conversation, C4 obtains customer basic profile from about dozen tables. Other tables are read and/or updated on demand.

**C4**

**Functional description**

C4 is an OLTP system that handles an interesting type of transactions. A transaction is initiated by a customer call, or so called Business Event. There are three types of events:

- Service Negotiations
- Account Management, and
- Trouble Call Management.

Each event is then divided into Tasks which are in turn broken into Activities. Tasks are groups of related activities that the representative and the customer have to complete. For example, if the negotiation is about the customer moving from one address to another, there will be a set of activities related to terminating some of the existing services, a set of activities related to obtaining the new address, and a set of activities related to negotiating new services and activation dates.

C4 must provide:
- Support for multiple concurrent tasks. For example, the customer should be able to negotiate a number of different services during the same call.
- Integrated support for completing activities (screen sequences, to-do list, context-sensitive data fields, etc.)
- Validation
  - of availability of requested service
  - completion of activities and tasks
  - integrity of customer data, and
  - integrity of the final requested configuration.
- Advice on available products and product “bundles”
- Resolution of conflicting events, and
- Support for interrupted and long-lasting conversations.

The last two points are particularly interesting. Resolution of conflicting event comes from multiple so-called Authors of events. For example, while a wife is negotiating a new phone line, the husband is using the phone company kiosk at his bank to request an ISDN line that will have an extra two phone lines. In general, events can come from:
- Direct conversations with company representatives (the case described here)
- Automated call center (the phone menu-type system)
- Kiosks (future), and
- Direct customer connections such as Internet (future).

Before C4 sends a service request to NOSS, it has to make sure that all related events have been combined or conflicts resolved.
The other requirement comes from the fact that a conversation with a customer can be interrupted (for technical reasons, for example) or suspended by the customer or the representative. The first case is rather obvious. An example of the second case when a customer says something like “let me talk to my wife and I will call you back”. In any case, C4 has to manage the context that persists and can be recalled.

Key architectural challenges

Here is a partial list of architectural challenges for C4. The challenges are not completely independent, so there is some repetition in the list. A number of challenges are implied by the execution architecture selected for the system. More about the architecture in the following section.

• Managing time and date effectively. Customer may want service changes in the future and this implies:
  • some form of a tickler system
  • ability to inform the customer about future changes of rates and/or services
• Interfacing with multiple authors of business events. As explained above, the main source of business events is direct conversation of an agent with a customer. However, other sources such as downstream systems, kiosks, etc. have to be accommodated. Important points:
  • business events from different authors may conflict with regards to the requested configuration at a service location
  • business events from different authors may be received and processed at the same point in time – business events from different authors become different service requests that must be cross validated to ensure a valid resulting configuration
• Architect something that is economically viable with a small set of customers and yet can grow to a very large network (i.e. 15 million customers)
  • it should not require high initial equipment investment
  • it should allow for “leaner” growth (in respect to cost(capacity) function)
  • it should be able to grow at a rapid rate (for example, 1000+ new customers a day)
  • identification, monitoring, and elimination of processing bottlenecks
• Validation of a requested service configuration should be done at near-real-time. This implies that:
  • C4 has to communicate with NOSS and other systems while guiding agents through tasks and activities, and
  • C4 may request to “lock” some of the network resources for a fixed amount of time (like a few phone numbers)
• Support for long-lasting, interrupted sessions. This issue has been described above
• Integrated (smart) performance support for company representatives
• Support a large number (e.g. 400+) of service representatives concurrently
  • a minimum of 100 service representative to start
• Near 7X24 application availability

Execution architecture

The system execution architecture is a standard three-tear C/S configuration as shown in Figure 2.
All agents workstations are PCs running the WinNT OS connected to a LAN that is interconnected to a WAN. The middle layer is a cluster of HP9000 servers running UNIX and a TP Monitor that can load-balance between them. The TCP/IP communications protocol is used throughout the network. The back-end runs on a dedicated high throughput LAN. This LAN connects the Enterprise and Billing DBMSs to the servers as well as a TCP/IP connection to NOSS via a module that marshals massages between C4 and NOSS. C4 runs on the cluster of PC and HP9000.

Additional architectural requirements are as follows:
- No persistent data caching on the agent workstations to limit the implications of local failures
- No DBs at office locations
  - no administrators at local offices
  - no maintenance down-time, etc.
- The middle layer server cluster tuned for performance
  - possibility to add serves to increase throughput
- The back end tuned for DB performance
  - preferred place to do persistency
- Well engineered operations architecture
- High availability cannot be achieved by utilizing fault-tolerant hardware (this option is not economically viable)
APPENDIX4: BOINC
(A case study)
Note: this is a simplified and generalized description of a real system

Introduction

The Berkeley Open Infrastructure for Network Computing (BOINC) is an open-source software for volunteer computing and grid computing. The primary function of the system is to make it possible for researchers to tap into the enormous processing power of personal computers around the world.

BOINC was originally developed to support the SETI@home project (http://setiathome.ssl.berkeley.edu), an experiment that analyzes radio telescope data from outside the Earth as an attempt to find extraterrestrial intelligence. The challenge was that the analysis of radio telescope signals required enormous amount of computation resource as the received signals consisted of much noise and man-made signals. The more computation resource involved, the wider frequency range coverage with more sensitivity\(^1\). Instead of adopting high-cost, supercomputers for the computation, a high performance distributed computing platform, BOINC, was developed to take advantage of the idling cycles of personal computers of people around the world who wanted to participate in the project.

Overtime, BOINC has become useful as a platform not just for searching for extraterrestrial intelligence but also for other distributed applications in areas as diverse as mathematics, medicine, molecular biology, climatology, environmental science, and astrophysics. It has about 320,121 active participants and 512,197 active computers (hosts) worldwide processing on average 6.6 petaFLOPS as of July 23, 2014.

The distributed nature and the gigantic scale of BOINC bring up several design challenges. For example, it must be able to manage a large pool of participating nodes (personal computers) remotely via computer networks. There could also be performance, security, and/or scalability concerns when designing its architecture. This is discussed in more detail later.

Volunteer Computing: How It Works

Participant’s Perspective

Volunteer computing performs computation using participants’ computation resource. Figure 1 depicts the participant’s perspective of a volunteer computing system. The following steps are indefinitely iterated:

1. A participating node (denoted as “Your PC” in the figure) receives a set of instructions,
2. The node downloads the executable applications and the input files for computation,
3. The node performs the computation and produces output files,
4. The node uploads the output files to the server, and
5. The node reports the list of completed tasks to receive reward and to adjust the size of future jobs.

Refer to http://setiathome.berkeley.edu/sah_about.php for more information.
Project’s Perspective

Volunteer computing provides an opportunity to projects that need large computation power and/or storage via collecting the resources from volunteers who wish to participate in the project. While the specifics of how a computation is performed differs system to system, but the high-level computational model (i.e., how the computation is divided, performed, and assimilated) is identical. Figure 2 depicts a volunteer computing system model in project’s perspective. The task server receives a computation as its input. The computation is divided into multiple computation tasks. A task is a part of the computation that is independent from the other tasks. Each task is duplicated into redundant jobs, which are the ones transferred in the second step of Figure 1. With redundancy, each task is executed as several identical jobs on distinct nodes. The created jobs are assigned to a randomly selected participant node via the job queue. When the nodes report back the results of the assigned jobs (step 4 of Figure 1), the task server compares and accepts the results. New jobs could be created based on the results. This process is iterated until no more jobs are left in the job queue.

![Figure 2: General Depiction of Volunteer Computing from the Project’s Perspective](image)

System Description

Overall System Description

BOINC has the general goal to nurture the public resource computing. It has the following specific goals:

- Reducing the barriers of entry to public-resource computing
- Sharing resources among autonomous projects
- Supporting diverse applications
- Rewarding participants

The Level 1 architecture (depicted in Figure 1) of BOINC is in the Client-Server architectural style. The architectural breakdowns of the BOINC client and server are in the following subsections.
BOINC Client

BOINC client consists of four major components and communicates with the components in BOINC server. Figure 3 depicts the BOINC client. The list of components is as following:

- The schedulers and data servers are installed on computers owned and managed by the projects to which the volunteers donate time of their computers.
- The core client communicates with the BOINC servers via the HTTP communications protocol to get and report work. The core client also runs and controls applications.
- Applications are the programs that do scientific computing. Several of them may run at the same time on a computer with more than one CPU.
- The GUI provides a graphical interface that lets the volunteers control the core client – for example, by telling it to suspend and resume applications. The GUI communicates with the core client via a TCP connection. Normally this is a local connection; however, it's possible to control a core client remotely.
- The screensaver runs when the participants are away from the computer. It communicates with the core client via local TCP, instructing it to tell one of the applications to generate screensaver graphics.

![Figure 3: High-Level Architecture of BOINC Client](image)

BOINC Server

BOINC server, including the task server, is depicted in Figure 4. It has three major components:

- Web interfaces for account and team management, message boards, and other features.
- Task server that creates tasks, dispatches them to clients, and processes returned tasks.
- Data server that downloads input files and executables, and that uploads output files.

![Figure 4: High-Level Architecture of BOINC server](image)
**BOINC Task Server Breakdown**

The BOINC task server must have *at least* the following eight major components:

- **The work generator** creates new jobs and their input files. For example, the SETI@home work generator reads digital tapes containing data from a radio telescope, divides this data into files, and creates jobs in the BOINC database. The work generator sleeps if the number of unsent jobs exceeds a threshold, limiting the amount of disk storage needed for input files.

- **The scheduler** handles requests from BOINC clients. Each request includes a description of the host, a list of completed jobs, and a request for additional work, expressed in terms of the time the work should take to complete. The reply includes a list of jobs and their corresponding tasks. Handling a request involves a number of database operations: reading and updating records for the user account and team, the host, and the various jobs and instances.

- **The feeder** streamlines the scheduler’s database access. It maintains a shared-memory segment containing (1) database tables such as applications, platforms, and application versions, and (2) a cache of unsent jobs. The scheduler finds jobs that can be sent to a particular client by scanning this memory segment.

- **The transitioner** examines tasks for which a state change has occurred (e.g., a completed job has been reported). Depending on the situation, it may generate new jobs, flag the task as having a permanent error, or trigger validation or assimilation of the task.

- **The validator** compares the instances of a job and selects a canonical instance representing the correct output. It determines the credit granted to users and hosts that return the correct output, and updates those database records.

- **The assimilator** handles job that are “completed”: i.e., that have a canonical instance or for which a permanent error has occurred. Handling a successfully completed job might involve writing outputs to an application database or archiving the output files.

- **The file deleter** deletes input and output files that are no longer needed.

- **The database purger** removes tasks and job database entries that are no longer needed, first writing them to XML log files. This bounds the size of these tables, so that they act as a working set rather than an archive. This allows database management operations (such as backups and schema changes) to be done quickly.

**Key Architectural Challenges**

Here is a partial list of architectural challenges of BOINC, the ones that must be considered for designing the BOINC task server. Note that the challenges are not completely independent; hence there could be some repetition in the list.

- BOINC must be resilient to erroneous computation results from nodes.
  - There could be malfunctioning computers or malicious nodes.
  - **Redundant computing** could be implemented to overcome this challenge. Replicate tasks into multiple jobs, distributed the jobs to random nodes, compare and accept a *canonical result* when the computation results are received from the nodes.

- BOINC server must be able to handle a large number BOINC clients.
  - It must be able to handle a large number of network connections with BOINC clients.

---

1 This is called the Traditional Redundancy. Refer to [4] for more information.
The BOINC task server has to be able to process a large number of simultaneous jobs performed by BOINC clients.

- BOINC server must be able to credit rewards to participants for successful computation results (e.g., the results that have been accepted as the canonical result).
  - The reward could be in the form of points and ranking between the participants.
  - The rewarding system must be able to vary the amount of credit based on the amount of resource (computation, storage, and network transfer) the participants provided.
  - The rewarding system must be able to work across different projects and applications.
- BOINC server must be able to adjust the size of jobs it assigns to BOINC clients based on the clients’ previous performance on the jobs that had been assigned to those clients.
- BOINC server must be available nearly 24/7.

References

This appendix is a summary of the following list of articles:

2. BOINC official website: http://boinc.berkeley.edu
3. BOINC wiki page: http://boinc.berkeley.edu/wiki/How_BOINC_works