Client-Server Architecture for Image-Based Photometry in the Astronomy Domain

Cohen Miller

Kennesaw State University

CS 4503—Parallel Distributed Computing

Section W01

Fall Semester 2023

***Abstract – Digital photometry is one of the most common techniques in modern-day astronomy. Massive amounts of photometric data are created every day, and researchers around the world are using machine learning and big data algorithms to develop efficient image processing schemes to process photometric images. The goal of this paper is to design a platform for these researchers that is capable of uploading, downloading, and managing these massive files with as little overhead as possible. A unique set of design considerations were developed that guided the design process, including the system architecture, and the more granular optimization techniques. The result of this development effort is a platform based on the client-server internet paradigm, that if implemented, would provide a responsive and dynamic system for processing image data using distributed computing.***

**Index Terms –** **Photometry, Astronomy, Distributed computing, Client-server systems, Image processing**

I. Introduction

**Photometry is a technique that involves measuring light energies with photosensitive instruments and is one of the most common techniques in modern-day astronomy. Digital photometric data can be created by handheld cameras, or the massive CCD arrays in modern day optical telescopes [1]. Massive amounts of photometric image data are flooding the scientific community, and the data requirements for modern-day photometric calculations are rapidly expanding. The Vera C. Rubin Observatory is an optical telescope system designed for real-time astronomical survey expected to generate up to 30 terabytes of data per night [2]. In the case of the Vara C. Rubin Observatory, this data is used to discover transients, which are astronomical entities that fluctuate in brightness over a period of time[3]. The applications of photometry are extensive, and a multitude of researchers are using machine learning and big-data algorithms to design and develop efficient image processing schemes to process photometric images. The purpose of this paper is to design a platform for these researchers that is capable of uploading, downloading, and managing these massive files with as little overhead as possible.**

**This platform is designed from the perspective of a client server paradigm, and the first section discusses the design structure and the design decisions that informed the architecture. The next section discusses the server-side platform in finer detail, particularly the specific optimization techniques that facilitate the high-speed performance that is required by time-domain astronomers. Section IV discusses the client-side interface and several tools for displaying and interacting with images in a real-time environment such as compression and selective image editing. In section V, the system’s set of protocols is discussed as well as the networking infrastructure that must exist for this system to fulfill its purpose. The final section discusses the author’s perspectives on the system, how the academic environment affected the software development process, and considerations for implementing the system.**

II. Client-Server Design/Architecture

**This section discusses** the client-server design structure and the design decisions that informed that architecture. This design was also guided by three considerations for efficiency in design for efficient processing in the astronomy domain, which are discussed in depth in sections II and III. The server-side architecture is designed in a layered form, where each component is designed to interact with the layers immediately above and below itself. There are five layers, the storage layer, the input/output layer, the servlet layer, and the HTTP server layer.

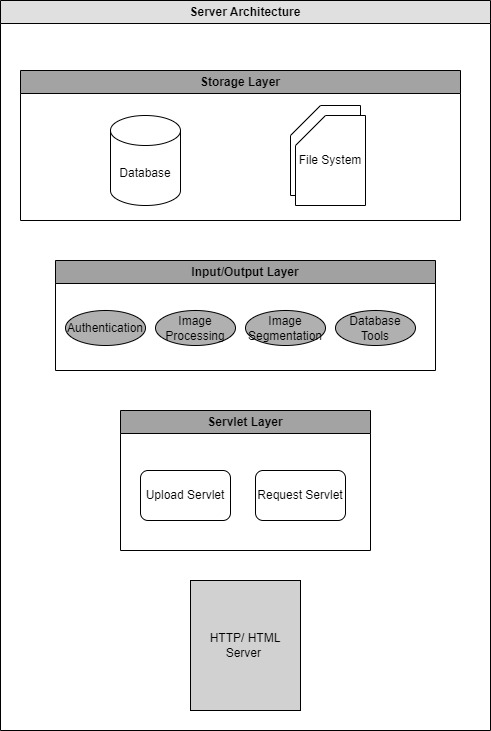


Fig. 1. Logical decomposition of the server architecture, particularly the four distinct layers: the storage layer, input/output layer, servlet layer, and the HTTP/HTML server.

A. *Server Side Design*

1) *Storage Layer:* The storage layer is responsible for maintaining all of the images, users, and their associated data. This is accomplished by two components that work in conjunction to provide an efficient storage structure: the database and the file system. The database serves two main purposes. The first is to store client information for validation and verification. The second is to store information about each image in the file system. The metadata for each image may include: the image owner, any data calculated by the system at the request of the user, and the location of the image file. The images themselves are not stored in the database, but rather in the server’s file directory.

*2) Input/Output Layer:* This layer serves several functions, but as a whole, serves to abstract certain low-level tasks like database access and image processing away from the servlet layer. It’s important to note that these components are not servlets and cannot be called independently of a servlet. It is also important to note that the image processing component is not actually a single function, but rather a library of actions that can be performed on an image. These actions are called at the discretion of the request servlet. In the context of the problem domain, these actions would range from “point-photometer”, which estimates the brightness of a region based on the contrast of the region relative to the sky, to “plate-solver”, which matches the image or images with known star patterns. These actions are obviously unique, but each has been designed to output data in logical “chunks”. The reasoning for this is explained in section II. The database tools component contains processes for interacting with the database. Servlets can use this component to add image data to the database, add user accounts for authentication, or query the database for image location or user information.

*3) Servlet Layer:* There are two main servlets in the servlet layer: upload, and request. The upload servlet is responsible for receiving files from the client. This servlet accepts an HTTP request from the client that contains file data, the file name, and an identity token from the client. This servlet first verifies the token, and then proceeds with the upload. If the token is correct, the servlet uploads the file to the file system, and writes the image path to the database in a new entry associated with the client’s name. In the event that the client uploads the image file for a specific purpose, certain image processing functions can also be associated with the upload request. In this case, the servlet will load the file, but will not write the file to the database or filesystem. Instead, it will redirect the request to the request servlet, and pass a pointer to the image in memory to the request servlet.

The request servlet is designed to coordinate image processing actions on images stored on the server filesystem. It may run when requested by the client, or when redirected by the upload servlet. When called by the client, this servlet accepts the name of the file to be processed, identity token, and the request to be executed. First, the appropriate image is located by querying the database using the file name and identity token. Next, the appropriate image processing action is selected and executed. As is outlined in the next section, the image processing functions are specially designed to process data incrementally, or “in chunks”. The request servlet uploads these chunks to the client as they are created. Once the whole image is created, the new image is written to the file system. This process is the same when the request servlet is called by the upload servlet, but the database is not queried.

*4) HTTP Server:* this component acts as the interface for servicing client requests. This component is responsible for uploading applets to the client and for coordinating uploads, downloads, and image processing requests. When a client requests a service, the HTPP server is responsible for calling the servlets stored in the servlet layer.

B. *Client-Side Architecture*

Applets represent a unique challenge for developing client-side architecture because they are designed to accommodate the user interface, application logic, the service logic in one component. This causes some difficulty when trying to modularize the system code. The solution this problem is to view each applet as a self-contained client system, each with its own presentation, application, and service layers. There are three applets in the client program, the image display applet, the data display applet, and the upload applet. Each of these applets communicate via an event-based system.

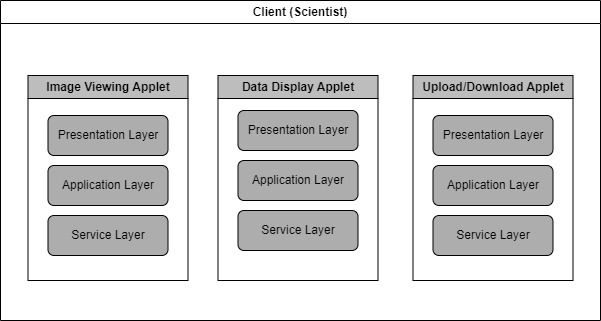


Fig. 1. The logical decomposition of the client architecture.

The compression-upload applet contains the GUI for the local filesystem. Users can browse their local filesystem and select files to upload to the server. This applet is designed in such a way so that images can be compressed and uploaded independently of the image display and the data display applets. Due to security concerns, an applet’s access to the local filesystem is usually highly restricted, so the compression-upload applet would be implemented as a privileged applet, which do not have the security restrictions of conventional sandbox applets [4].

The tools-and-data applet contains the graphics display logic for data-based tools. When a tool is selected, the application and service layers instantiate a connection with the client by creating a WebSocket object. If the request involves an image transfer from the server, the application layer notifies the image display applet and passes the WebSocket object so that the image display applet can receive the image file.

The presentation layer of the image display applet contains the logic to display the images on the local device and those that have been downloaded from the server. The application layer accepts the WebSocket from the data display applet and the service layer listens for a response from the client that contains the image files. One critical detail of this applet is that it must display the file *as it is downloaded*.

C. *Server Design Decisions*

Most servers can be categorized based on how they handle client requests. Servers can be concurrent or iterative, stateful or stateless, and connection-oriented or connectionless. These classifications are important because they dictate the nature of both the client and server architectures. The first classification, concurrency was specified in the design requirements of the system. This obviously produces an incentive for a large number of logical processors to increase the capability for parallelization. This server also uses a hybrid of global and session state storage. Global state information is used for client validation and verification, and for associating clients with their data, while session state information is used to maintain client information during a session. This is exemplified in the specifications of the upload and request servlets. Finally, the server uses a connection-oriented paradigm. This decision was made because of the accuracy requirements for astronomy images, and will be discussed in greater detail in section V.

III. Server-Side Platform Optimizations

In the astronomy domain, data sets are notoriously large. This is due to the need for high image accuracy and the ability to capture entire hemispherical images. Even single images may display the entire horizon, and astronomical endeavors such as the Gravitational-wave Optical Transient Observer (GOTO) involve arrays of optical telescopes taking hundreds or even thousands of images each day to conduct all sky transient surveys or monitor astronomical patterns [5].

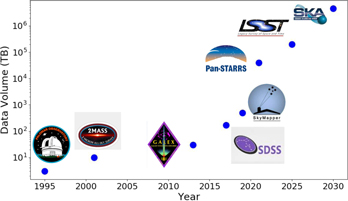


Fig. 3. The progression of the yearly data volumes over the past two decades. and an estimated yearly volume of over 106 terabytes of data by the year 2030. [6]

The primary challenge when building this client server system is handling the sheer amount of data. The author of this paper created some considerations early on in the design process that would establish a framework to approach optimization problems for images of such magnitude. These design considerations can be likened to the non-functional requirements of the software engineering discipline in that they inform many of the design decisions throughout the software development life cycle. The following sub-sections introduce each of these design considerations and explain how they were used to optimize the system.

A. *Maximize Processing Resources*

This consideration deals with utilizing computing resources to their fullest potential under various conditions and usage levels. The idea is that providing the server with multiple options for parallelism will theoretically improve performance under every possible condition. For example, if the server is experiencing a high volume of client requests, the server can create many threads to concurrently service those requests. On the other hand, if there’s a small number of *computationally intense* requests, the server can utilize multiple threads *per request.* In the server architecture, parallelism can occur at the client level simply by the nature of the servlet architecture. When a client sends an image processing request via HTTP, the servlet container receives the request and creates a new thread to handle the request. This thread uses the servlet to fulfill the request. Parallelism can also be implemented at the data level. This is implemented in the server-side architecture and is accomplished by the image processing components, which are designed to work in a series of chunks, and massive images can be subdivided into their constituent parts and be processed piece by piece.

B. *Distribute Networking Tasks*

In this particular domain, poor upload and download implementation could effectively cripple an otherwise “speedy” solution. Let’s say a massive file is being processed. Normally the image would wait to be fully assembled before being sent to the client. Meanwhile, the client and the network hardware sit idle. When the image is fully assembled, the entire image must be sent at once. The right combination of high-volume requests could result in a severe bandwidth bottleneck. However, if the data could be sent to the client incrementally, this would distribute the network load over time. Because of the data parallelism implemented in the server-side architecture, we are given a unique opportunity to send data to the client *as it is created*. This should theoretically reduce potential bottlenecks in addition to reducing client idle time. Table I illustrates a potential pseudocode solution that pushes the image bytes to the network as soon as they are received.

Table I

Sending Images Incrementally

|  |
| --- |
| /\*  \* Pseudocode Algorithm describing incremental  \* Downloads to client inside request servlet  \* Author: Cohen Miller  \*/  // SERVLET CODE  imageProcessingService.begin()  OutputStream outputStream = response.getOutputStream()  //process image piece by piece  while (imageProcessingService is NOT finished) {  byte[] chunk =.processChunk()  outputStream.write(chunk)  outputStream.flush();  } |

C. *Minimize Uploads, Downloads, Reads*

Upload and download times are limited by bandwidth, geographic location, and packet loss. Due to the decision to use TCP based connections, packet overhead is also a limiting factor. These factors are exacerbated by the size of astronomical images. Similarly, the size of these images means that even reading the files from the filesystem into RAM can be a time-consuming process. Thus, two measures must be taken to optimize these types of data transfers: The number of times the data must be transferred must be kept to an absolute minimum, and the amount of data transferred must be the least possible amount to accomplish the task at hand. One good example of implementing this design consideration is for a function like point-based photometry. Virtual photometers estimate the brightness of a region (such as a star) based on the brightness of that region relative to its surrounding pixels (the sky). Instead of loading the entire image into RAM, the image segmentation tools can be used to only load the point and a small section of the surrounding area. Another promising method for reducing transfer loads is the use of lossless compression. Ideally, this would be executed on the client side, and is discussed in further detail in section IV.

IV. Client-Side Image Display Optimizations

Before discussing the different optimizations that have been implemented to improve the client-side performance, a brief discussion of the interface operation is in order. The client-side system has four fundamental requirements for operation. First, users must be able to upload sets of images as well as individual images to the server. Second, users must request different types of data calculations that are based on the images stored on the server. Third, the results of the data calculations must be retrieved and displayed. Fourth, users may request image processing calculations that return new image files to the client side for display. This system was explicitly designed such that heavyweight processing is accomplished server-side. The client receives data of two distinct types: real-time text-based data, and static image files. Consequently, the majority of the responsiveness of the client side depends heavily on the performance of the server. There are two primary optimization methods that occur specifically on the client side, namely compression, and parallel image display. These two techniques were carefully applied based on the considerations for design outlined in the previous section.

A*. Client-Side Compression*

Fpack is a suite of lossless compression and decompression tools published by NASA to handle photometry images [7]. These tools are capable of compressing files in the FITS Format, which is the most popular format for storing astronomical images. The Fpack suite was selected for this system because its utilities are designed to utilize parallel processing and typically outperform other common compression tools such as GZIP. Compression is initiated on the client side for three reasons. The primary reason is because this results in more efficient upload and download bandwidth usage. Second, operating on a smaller file improves the server-side performance and reduces storage requirements in both hard storage and memory. The third reason is so that clients can customize their compression requirements. Fpack is designed with several customization parameters such as compression ratio and dithering [8]. Users can select these customization parameters per their own discretion before uploading to the server.

B. *Spatial Image Edit*

The image display applet is the most computationally demanding component on the client side due to the high resolutions of the image files that it displays. This applet carefully aligns with the design considerations discussed in section III by using an incremental approach to display files piece by piece. This incremental approach makes a selective image editing system possible. If a user desires image processing only on specific spatial regions of the file, a special request is sent to the server that includes the type of effect, the image to be processed, and targeted area of the image. The server-side request servlet processes this information and returns the image snippet instead of the whole file. The image display applet dynamically displays the new section using the incremental display system. This way, the client avoids downloading the entire image for a second time.

V. Assumptions about Network, Protocols, Operating system Tools

This section discusses several assumptions about the network infrastructure and explains decisions about the decision-making process behind the protocols implemented in our system’s design.

A. *Assumptions about Network*

There is one challenge for implementing this sort of TCP/IP based computational system that has not been addressed, and that cannot be addressed within the scope of this paper. That is the limitations of the networking infrastructure. Most standard internet connections are not equipped to handle the sheer magnitude of data that would be required for this client-server system to operate in real-time. At the time of project completion, the Vera C. Rubin observatory mentioned in the introduction will utilize two fiber optic connections that operate at a rate of no less than 100 gigabytes per second [9]. The system presented in this paper is obviously designed with speed and scalability in mind, but without a reliable high speed internet infrastructure, bandwidth limitations would end any hopes of a responsive system.

B*. Choice of Protocols*

Several different configurations of internet transfer protocols were considered during the design process. Ultimately, the TCP-based WebSocket protocol was selected. At a high level, the decision between TCP and UDP was carefully considered. UDP is known to excel in real time systems because the protocol performs with much less processing overhead than TCP. On the other hand, TCP is the industry standard for transmitting images with high reliability over network connections. It has more well-known tools and protocols like HTTP. Ultimately, the unreliability of UDP proved too great of a liability, owing to the fact that the majority of photometry use cases require extreme precision. Time domain astronomy in particular studies celestial events that occur in a specific period of time known as transient events. The rate and duration of transients is empirically estimated, and the capture frequency of the imaging hardware is set to the largest interval between frames that can be reasonably assumed to capture the transient event [10]. This means that even individual images are of the utmost importance, with the understanding that the loss of a single frame or even a segment of a frame due to packet loss could compromise the study. Having selected TCP, the next decision was whether we should utilize existing HTTP based systems or implement our own custom sockets. HTTP is a common protocol that favors security, while custom sockets allow greater control over non-text transfers (like image files). A hybrid of the two was even considered at one point, where HTTP could be used for most of the user requests, while image uploads would be done using custom sockets. After some research, the WebSocket protocol was selected because of its mutual support of HTTP requests and traditional TCP byte streaming. This allows the system to use the familiar HTTP/HTTPS framework for most user requests and then “upgrade” the connection to use byte streaming for transmitting large files without the HTTP overhead [11].

VI. Perspectives About System and Implementation Ideas

A. *Perspectives About System*

This system was developed in an academic environment, and there are several differences from how the server would’ve been designed in a professional setting. The most notable difference is that the problem domain, astronomy, was chosen to contextualize the initial concept of an image processing server, while in a professional setting, the domain would dictate the server architecture, rather than the other way around. This system was not designed with any concrete functional or non-functional requirements in mind, and so I spent a lot of time deciding what set of problems the server should solve. This assignment encapsulated the “design” portion of the software development life cycle, while in a professional setting, the entire software development methodology would be followed, including the planning and requirements management phases. The software tools that were chosen to implement this system are also based on the concepts studied in the class, such as statement level and unit level parallelism, servlets, and applets. As a result, while I tried to be as factually correct as possible, I was essentially trying to design a professional level system using software tools that are nearly as old as I am. That is not to say that these tools are not the tools for the job, in fact, servlets seem to suit the problem domain well. Also, due to the academic nature of the paper, several real-world components were omitted from the design. Most notably that of user security and validation. For a large portion of the design process a third servlet, which I called the authentication servlet, was responsible for validating the client. This servlet would use authentication and database functions stored in the input/output layer to create a session token that the client would use to verify its identity throughout the session. This servlet wasn’t included in the final architecture because it didn’t contribute to the fundamental problem of an image processing server.

B. *Implementation Ideas*

If this system were to be implemented, it would require extensive testing. There are a multitude of alternative architectures and fine adjustments that can be made that may potentially improve the system’s performance. These concepts were not included in sections III and IV because there wasn’t enough data to determine whether they would definitively improve performance. For example, while incremental uploads and downloads were implemented in the final solution, parallelizing the upload/download process was also considered. I also learned about content delivery networks while studying for this assignment. They are essentially networks of proxy servers that store frequently requested files. They can reduce network latency when clients and the server are geographically distributed. While this is often the case (telescopes are frequently located far away from civilization to reduce light pollution) a CDN would be infeasible because most processes involve direct communication with the server for image uploads and data requests. However, a CDN may improve performance with applet downloads.

VII. Conclusion

The various sections of this paper outlined a server system that, if implemented, would serve as a platform for astronomical survey and analysis using image-based photometry. Section II covers the architecture of the server and client-side systems, and discussed the layered approach that facilitates efficient parallel processing. Section III covered the three considerations for efficiency in design that drove the design of the server system: maximize processing power under every scenario, distribute upload and download processes to reduce bottlenecks, and minimize unnecessary uploads, downloads and reads. The paper then went on to discuss the client-side optimizations such as compression and parallel image display, and this section also proposed a technique for spatial image editing. Section V discussed the network infrastructure that must be present to support the system, and the decision-making process behind the use of the WebSocket protocol. The final section reflects on the development process and acknowledges the differences between the academic environment in which this system was designed, and the professional setting in which it would theoretically be deployed. This section concluded with several techniques for optimizations that weren’t implemented in the final design.

Distributing computing has seen widespread use for over two decades, but even now is rapidly growing, and as data processing systems become more powerful and more capable, the domains that utilize those systems rapidly expand as well. The keys to solving some of the challenges of time-domain astronomy lie within modern server design and processing techniques. Although this platform was conceived by an individual outside of the domain of astronomy, the claims made in this paper are based on a substantial amount of scientific literature and research. It is from this perspective as an outside observer that I attempted to solve some of those challenges that appear most imminent, namely the storage, processing, and transmission of high-resolution image files. An individual within the domain of may have engineered a more accurate set of functional requirements and a more applicable set of problems to solve. Similarly, at this point in time, I lack an in-depth understanding of modern-day server design, and very little modern literature about server design was reviewed when designing this system. As a result, there may be existing conventions that solve the issue of data transfer in a way that renders certain elements of this design obsolete or archaic. However, it is my hope that some of the techniques described in this paper, although basic, will provide a unique perspective on addressing the challenges inherent in processing such vast astronomical datasets.

References

Hewett, P. C., Warren, S. J., Leggett, S. K., & Hodgkin, S. T. (2006). The UKIRT Infrared Deep Sky Survey ZY JHK photometric system: passbands and synthetic colours. Monthly Notices of the Royal Astronomical Society, 367(2), 454–468. <https://doi.org/10.1111/j.1365-2966.2005.09969.x>

Stephens, M. (2008, October 3). Mapping the universe at 30 Terabytes a night • The Register. Retrieved November 11, 2023, from <https://www.theregister.com/Print/2008/10/03/lsst_jeff_kantor/>

Kremer, Jan, et al. “Big Universe, Big Data: Machine Learning and Image ... - IEEE Xplore.” *Big Universe, Big Data: Machine Learning and Image Analysis for Astronomy*, IEEE, Mar. 2017, ieeexplore.ieee.org/document/7887648.

What Applets Can and Cannot Do (The JavaTM Tutorials >Deployment > Java Applets). (n.d.). <https://docs.oracle.com/javase/tutorial/deployment/applet/security.html>

Dyer, M., Steeghs, et. al (2020). The Gravitational-wave Optical Transient Observer (GOTO). https://doi.org/10.1117/12.2561008

Maireles-González, Ò., Bartrina-Rapesta, J., Hernández-Cabronero, M., & Serra-Sagristà, J. (2023). Efficient lossless compression of integer astronomical data. Publications of the Astronomical Society of the Pacific, 135(1051), 094502. <https://doi.org/10.1088/1538-3873/acf6e0>

Pence, W. D., White, R. L., & Seaman, R. (2011). FPack and Funpack User’s Guide: FITS Image Compression Utilities. arXiv (Cornell University). <https://arxiv.org/pdf/1112.2671>

Vassi. (n.d.). Amlight-Exp Activates two new 100 Gbps Points-of-Presence Enhancing Infrastructure for Research and Education. Research Education Collaboration. <https://www.amlight.net/?p=3055>

Bellm, E. C. (2016). Volumetric Survey speed: a figure of merit for transient surveys. Publications of the Astronomical Society of the Pacific, 128(966), 084501. <https://doi.org/10.1088/1538-3873/128/966/084501>

Sookocheff, K. (2019, April 4). How do websockets work? Kevin Sookocheff. https://sookocheff.com/post/networking/how-do-websockets-work/

Barchi, P. H. et al. (2020). Machine and Deep Learning applied to galaxy morphology - A comparative study. Astronomy and Computing, 30, 100334. https://doi.org/10.1016/j.ascom.2019.100334

Liu, M. L. (2004). Distributed computing: Principles and Applications. Addison-Wesley. Maireles-González, Ò., Bartrina-Rapesta, J., Hernández-Cabronero, M., & Serra-Sagristà, J. (2023).