Chapter 6

Experimental Control and Logging Database

This chapter describes the implementation of a relational database management system (RDBMS) and associated databases into the experimental control and data-logging flow of our experiment. Changes to existing applications in the Cicero word generator software suite and new applications that were developed to interface with the database are also described in detail.

6.1 Motivation

Most quantum gas experiments generically run in a cycle: atoms are trapped and cooled to degeneracy, some experimental parameter is varied compared to the previous run, and an image or series of images are taken with one or more scientific cameras. Saved images can then be imported into scientific programming suites for analysis. Building off of this, various labs have written various custom applications which perform lab-specific tasks to meet their data acquisition needs; for example, a routine might poll a folder for new image files and automatically load any new files for analysis as they are saved, or an application may save the value of some experimental parameter that is being varied into a running text file. Often, there has been redundant/parallel development of solutions for different experiments, and many of

the custom solutions need to be heavily modified or replaced when the needs of the lab change.

The Cicero software suite is also limited in its automated running capabilities: in its original form, experimental parameters could only be varied in predetermined lists. As quantum gas experiments have become more complex, there has been a growing interest in the utilization of search and optimization algorithms (such as genetic algorithms) to minimize the cycle time of experiments, maximize some signal of interest, or even to determine information about the phase diagram of a system [58, 59].

The goal of developing a database backend to the Cicero software suite - and some new applications to interface with it - was thus manyfold:

- Automating the storage of all experimental (and eventually, environmental) parameters during each run of the experiment to allow for searching and sorting in high-level ways.

- Creating a system where the same set of simple platform-independent commands (i.e., SQL commands) could be used to access laboratory data from any computer or application.

- Enabling the existing experimental control software to be able to run search and optimization algorithms by allowing Cicero to interface generically with outside applications

- Developing new protections to enable reliable automated operation of Cicero, including rigorous synchronization between the experimental parameters and images and the ability to interlock on external conditions

6.2 Summary of Developments

The software development related to the implementation of the database consisted of several key parts:

- The definition of a database schema which does not contain redundant or inter-dependent columns, and which is capable of being easily modified as the needs and scope of the experiment change
- The development of a database-interfacing application called Zeus which serves as a "traffic director" for the control of the experiment.
- The development of a USB-interfaced laboratory hardware monitor, which allows for Zeus to determine whether the experimental sequence should run or be paused based on external conditions such as measured laser powers.
- The development of a C# API which allows for Window-based applications to quickly access all of the images, processed data, and experimental parameters stored in the database, without requiring detailed knowledge of the internal structure of the database.

In addition to these primary developments, a few other pieces of hardware and software have been developed which demonstrate the expanded capabilities of the experimental control system. These include two genetic algorithm-based optimization applications which can optimize experimentally measured quantities (e.g., total atom count or condensate fraction), and a variety of wirelessly interfaced sensors which open the door to both the idea of "big data" logging of environmental variables and flexible software-based interlocking schemes to protect the experiment during long stretched of automated running.

6.3 The Database

Because the Cicero suite of laboratory control applications was written for the Windows-specific .NET framework, and because many pieces of hardware used in the laboratory (such as cameras) only have Windows-compatible drivers, all of our laboratory control computers run versions of the Windows operating system.

In contrast, RDBMS software is arguably the most straightforward to set up, back up, and maintain when it is running in a Linux environment. We thus set up a central

data logging server running an Ubuntu distribution of Linux, and set up the MariaDB RDBMS on it. MariaDB is a fork of the more popular MySQL software, sharing all of its essential characteristics.

Figure 6-1 shows all of the computers used in running the dysprosium experiment, which communicate with one another through the gigabit ethernet-compatible LAN.

6.3.1 Structure of the Database

The database is divided into several tables, each of which has a column-row structure, with columns corresponding to data fields and rows corresponding to individual saved entries in the table. One column of each table serves as a unique integer, called the table's primary key, which is used to give a unique identifier to each row.

The tables used are the CiceroOut, Images, Cameras, InputValues, Updates, Sequence, and Analysis tables. The layout of the database is shown in Figure 6-2.

The CiceroOut table stores the values of all of the variables defined within the Cicero word generator suite for every run of the experiment; these variables define the laser powers, laser detunings, and power supply currents during each run, as well as durations that are varied. The timestamp of the run and any user-entered notes about the run are also stored in the database. Because the variables values are stored as numerical values and can thus be used in the construction of comparison or logical SQL commands, this allows for streamlined searching through the database for experimental runs meeting certain criteria. For example, one can easily construct a command which allows them to search for the ID numbers of all runs for which the detuning of an imaging laser was negative:

```
SELECT runID FROM CiceroOut where ImagingDetuning < 0;
```

The Images table stores the raw binary data of each image acquired by a camera during the experiment. In each row there are up to three images: Atoms, Light, and Dark. For a typical absorption image, all three images are taken, and are then saved to be later used to construct the composite image from which the total atom number and fits of the cloud profile can be extracted. Along with each entry in this table

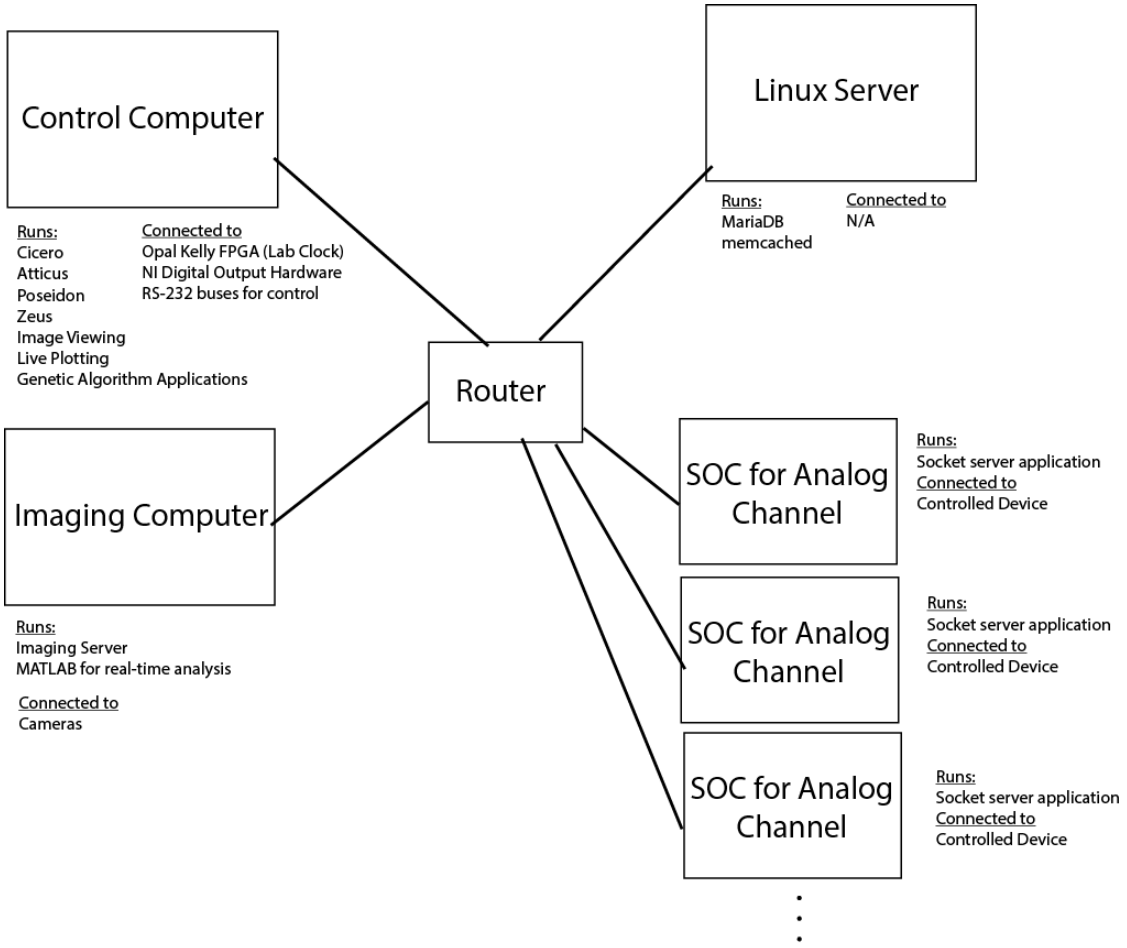

Figure 6-1: The network-enabled devices used in our experiment. Most control software runs on a single computer. The Poseidon server discussed in Chapter 5 passes buffer information to each of the SOCs over the network connection as previously described. The Zeus server, memcached daemon, and genetic algorithm applications noted in the diagram will be discussed later in this chapter.

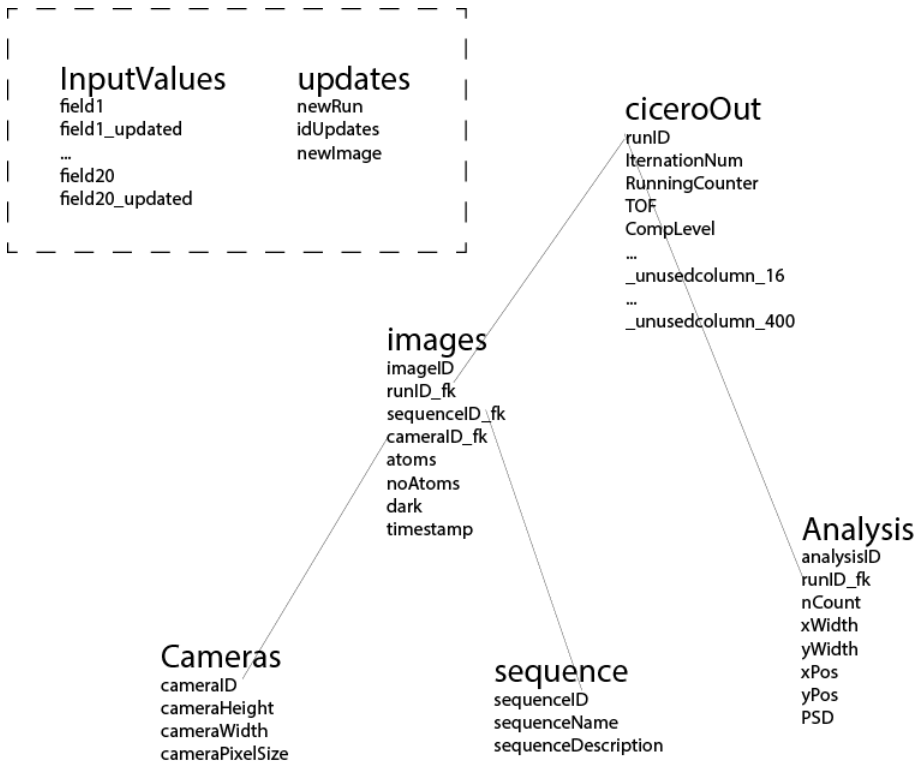

Figure 6-2: The tables and fields of the experimental control/logging database. The large text indicates the name of a table and the small text below it indicates the names of the fields belonging to that table. The two tables in the dashed box always contain exactly one row of data, which is updated by various applications during the experimental sequence. The lines indicate relationships between primary keys and foreign keys, as described in the text.

is stored the CameraID of the camera which acquired the images (see the following paragraph), the runID of the associated experimental sequence run by Cicero, and a unique ImageID primary key field. A database call which can pull up all images associated with specific experimental criteria is easy to construct:

```
SELECT runID FROM CiceroOut where ImagingDetuning < 0;
SELECT imageID FROM Images where runID_fk = $result;
```

where the "$result" variable represents the result of the first line. The abbreviation "fk" here refers to the fact that in the images table, the runID refers to a field in another table (CiceroOut) which is the primary key for that table; the runID_fk field in Images is thus a "foreign key" while the runID field in Cicero is a primary key.

The Cameras table stores the specifications of each camera (or special mode of use of each camera) in the lab. These specifications include the sensor dimensions and the physical pixel size, and can be called to correctly reconstruct the images stored in the Images table and to obtain the correct measured atom number from them; images must stored in the Images table as 1D arrays, and so must be reshaped using the saved values of the camera's sensor dimensions. For example, the commands to determine the sensor dimensions for a particular image might look like this:

```
SELECT cameraID_fk from Images where ImageID = 16234;
SELECT width,height from Cameras where cameraID = $result
```

The InputValues table allows for Cicero variables to be bound to values which are controlled by external applications which have access to the database. Prior to the implementation of this database table, the values of experimental control variables in Cicero could only be specified prior to a batch of runs commencing, precluding the implementation of search or optimization algorithms. The InputValues table consists of a fixed number (twenty in our case) of rows – although more can be added easily – each of which stores the value to be associated with a variable, and a field indicating whether the value has been updated for the next run of the experimental sequence. Using this table consists of binding a variable in Cicero to one of the rows of this table, and then having an external application (such as a search/optimization

algorithm application) update all of the bound fields each run. Since it may take several seconds to e.g. fit the detected cloud shape after an image is taken, the field indicating whether the bound fields have been updated can be used by the Zeus server to halt the execution of the next experimental sequence until after the fields have been updated.

The Updates table is used to store some simple binary variables which are used as flags to maintain synchronization between applications interfacing with the database. The "newRun" field is set to true whenever a new experimental sequence begins, alerting any data acquisition applications that the data they are about to collect is associated with the most recent runID value that can be found in the CiceroOut table. The "newImage" field alerts Zeus that new images have been collected. As will be described in the following section on Zeus, images are initially saved into a RAM-only cache on the Linux server to facilitate immediate viewing and/or analysis, but only permanently saved if the user has opted to save all images to the database. The newImage field tells Zeus that new image data is ready to be saved to the database if this has been enabled.

The Sequence table stores the name of and short description of each experimental sequence, as specified by the user within the Cicero control software. This allows for convenient string-based searching of the database for specific sequenceIDs, which are then associated through the Images table with any saved data and the experimental parameters in the CiceroOut table.

Oftentimes, image analysis is set up to be performed immediately as images are acquired, but then images are later loaded and reanalyzed more carefully later. The results of the real-time analysis are stored in a table called Analysis, which associates the measured atom number, parameters of functional fits, and composite quantities derived from these values with a specific runID (as a foreign key in the table). The reason that the values are associated with a runID and not an imageID is that one can elect to save hard drive space by not saving any of the image data during a long batch run, but only saving the real-time analysis results; in this case, there are no imageIDs in the database to associate with the analysis results.

6.4 The Zeus Server

The additional functionality made possible by the implementation of the database backend, as well as the inevitable increase in complexity, warranted the development of an all-in-one application to streamline the usage of the database. Zeus is an application which performs a variety of auxiliary tasks involving communication with and through the database. The tasks performed by Zeus are

1. Saving the experimental variable values and timestamp to the CiceroOut table during each run of the experiment

2. Ensuring that acquired images have been associated with the correct runID by only allowing a new run to begin after all images have been acquired. This especially prevents hang-ups or lag in the data acquisition software from causing images to be erroneously associated with the *next* run's entry in the CiceroOut table.

3. Monitoring user-specified hardware and software conditions to determine whether the experimental sequence should run or be paused

4. Saving captured image data, initially stored in a temporary RAM cache for fast access, into the Images table if the user has specified this

5. Ensuring that if Cicero's variables are bound to fields in the InputValues table, Cicero does not run until these fields have been updated by external applications

The connection between Cicero and Zeus follows the same model as the connection between Cicero and the hardware control server Atticus; the connection is made using .NET remoting, wherein Cicero (the client) passes data to Zeus (the server) and remotely executes functions defined on Zeus. Most of Zeus's functionality is thus controlled by Cicero, with checkboxes and input fields in the main Windows form primarily allowing users to specify options.

6.4.1 Data Flow

A complete run of the experiment, using the database and the Zeus application, proceeds as follows.

When an experimental sequence begins, Cicero first runs a function called "CheckIfCiceroCanRun" on Zeus.[1] Zeus then checks for various conditions to be met, based on the user-specified options regarding software and hardware conditions, and whether the "manual override" checkbox has been clicked to prevent the sequence from running. Examples of these conditions include whether a specific piece of software is still running (monitored by ensuring that Zeus is receiving a regular "heartbeat" from that application, i.e. a new random number every second or so) or whether a digital voltage line in the lab is reading logical high.

Once all conditions have been met and the manual override box has been unchecked, Zeus checks to see whether any variables which have been bound to the database are still waiting for updated values. This may be the case if the analysis of the previous image has not yet completed, or if an optimization application is still computing what the next set of variable values to try should be. As soon as each bound variable has received an updated value, the variable values are moved into Cicero's application memory and the database fields are reset to their non-updated states in preparation for the next run.

Once any bound variables have been updated, Cicero passes all of the information about the experimental sequence to Zeus, and Zeus saves all of the variable values, a timestamp, and the user-entered description of the sequence to the CiceroOut and Sequence tables. If new variables have been added since the last time data was saved to the CiceroOut table, Zeus modifies the structure of the table and associates a new column with the new variable.

The experimental sequence then runs as specified by Cicero and carried out by

[1]Technically, all functions called over .NET remoting by Cicero are run on all connected servers, including the hardware control servers Atticus and Poseidon. All of these servers must have a definition for all functions that Cicero will call, but the functions can simply be set to return "true" if that server is not relevant to that function call. For example, the CheckIfCiceroCanRun function is defined in Atticus and Poseidon, but simply returns "true" when called.

hardware servers Atticus and Zeus. At the end of the sequence, data acquisition software (typically camera control software) is triggered and one or more images is acquired. These images are then saved immediately into a RAM-only memory cache on the Linux server. The motivation behind this step is to move the image data out of program memory and into a place where other applications in the lab can read it immediately, but prevent hard drive reading/writing times from adding additional time to the experimental sequence. For example, the routine determination of the resonance frequency of a transition involves taking many absorption images and comparing the optical depths at different frequencies; there is no need to save any of these images as soon as the OD is extracted, however, and so it is sufficient to have the images be analyzed in real time as they are moved into the cache.

Once image data has been acquired and stored in the cache, Zeus will move the image data from the cache to the database for permanent storage if the user has elected for this. In this case Zeus will also halt the execution of the next experimental sequence until the image data has been saved, in order to ensure that the image acquisition software associated the image data with the correct runID, and to prevent the sequence from repeating more quickly than image data can be saved (in the case of very short sequences of around a second in duration).

6.4.2 External Hardware Monitoring

A well-built quantum gas experiment can perform hundreds or thousands of experimental sequences in a row without any human attention. Inevitably, certain things will degrade over the course of a day or night, such as the fiber coupling efficiency of trapping lasers or the alignment of a sensitive beam. While hardware interlocks will usually be set up in the lab to prevent degradations like these from causing damage to the equipment, there was no functionality in place to prevent the continued execution of the experimental sequences (and acquisition of meaningless data) if some hardware condition or conditions were not met.

To address this limitation, a small microcontroller-based circuit which can measure several digital and analog voltages simultaneously, and communicate their values