Passivity Based Control in Software Systems

Example and Case Studies

Kashev Dalmia[1], Tarek Abdelzaher[1], and Shen Li[1]
[1]University of Illinois at Urbana-Champaign

ABSTRACT

In this paper, we use passivity theory as an approach for dealing with dynamical systems, and demonstrate how to apply it to software systems in a general way. We first cover key results from passivity theory. Then, using an example, simulated system, we demonstrate how to design a controller which guarantees asymptotic BIBO stability for a system using a passivity based control, or PBC, approach. Finally, we examine more complex spftware systems from other publications, Proteus and Pyro, and demonstrate how to apply passivity theory to these kinds of systems.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Process control systems; I.2.8 [**Problem Solving, Control Methods, and Search**]: Control theory

General Terms

Software, Control, Passivity

1. INTRODUCTION

Though control theory, and specifically, passivity theory, has been around in the control community for many decades, it is taking software engineers a lot longer to pick up on these principles in their work. Usually when an average software engineer comes across a problem like load balancing, at best, they will recognize it as a control problem, and design a control loop, and implement a linear type controller to solve this problem. At worst, they will come up with something much worse. And this is not even necessarily the best solution!

Software in general is very non-linear. Because of this, linearization and control via PID or a similar method does not always work well. In this paper, we introduce a framework for thinking about software in terms of *passivity*. Passive systems are, in general, systems which do not produce their own energy. Software components are not like physical systems in that they do not necessarily have an obvious notion of energy, but nevertheless, they can be analyzed as passive just the same. Once a system has conditions under which it is passive, then this can aid greatly in control design.

First, we review the necessary building blocks and results from passivity theory in Section 2. Then, we present an example system, with analysis and results in Section 3. Finally, we examine some real, published software systems (Proteus and Pyro) and examine how passivity theory can be used to analyze their control in Section 4.

2. PASSIVITY THEORY

Control of systems is an increasingly important area of research. Traditional control methods rely on linearization of a system around a control point. However, many real life systems are not actually linear. The next class of controllable systems which are possible to reason about (and as such, are the basis of modern control theory) are so-called *passive* systems.

The term *passive* arises from the context of electrical circuit analysis. Informally, passive systems are those for which a notion of virtual "stored energy" can be defined, such that the change in stored energy is never greater than the energy supplied to the system externally. In this section, we review the definition and conditions for systems to be considered passive, and importantly, the implications this has for the stability of such systems. Specifically, passivity of systems has implications for the passivity and stability of their interconnections, which makes it easier to construct useful, stable software systems.

The following section reviews basic passivity theory useful for reasoning about software systems. Readers interested in further details, including proofs and a more rigorous treatment of this section, are encouraged to see [2, 13, 8].

Note that in many cases in this section, we present multiple input, multiple output formulations of these definitions and properties. Thus, inputs, outputs and states will be considered to be vectors. However, we assume that the number of inputs and outputs to the system are equal in number. It is easy to translate these properties to single input, single output properties that we will use more in later sections.

2.1 Passivity of Systems

A general system component will have a few different components: an input $u(t)$, an internal state $x(t)$, and an out-

put $y(t)$ (where u, x, and y may, in general, be vectors). The component model describes how the internal state and output of the component evolve over time, and how these components interact. For a general, non-linear system:

$$\dot{x}(t) = f\left(x(t), u(t)\right)$$
$$y(t) = h(x(t), u(t)) \tag{1}$$

where f and h are, in general, non-linear functions. We will use this general system throughout this section.

Passivity theory defines a *supply rate*, $w(t)$, as a general function of the input and output of the system:

$$w(t) = w\left(u(t), y(t)\right) \tag{2}$$

This supply rate represents the "virtual energy" supplied to the system. Systems then in turn store this energy. The *storage function* is denoted as $S(x)$, and is on all of the system state $x(t)$.

A system is considered to be *dissipative* if there exists a non-negative and real $S(x)$ such that

$$S(x_1) - S(x_0) \leq \int_{t_0}^{t_1} w(t)d(t) \tag{3}$$

for all $0 \leq t_0 \leq t_1$, where x_n is the state of the system at t_n with a given starting state x_0 and an input u. If the storage function is differentiable, which it almost always shall be since we must come up with it, then the equation can be written in a derivative form:

$$\frac{dS\left(x(t)\right)}{dt} \leq w(t) \tag{4}$$

This is a precise, mathematical way of saying that the rate of change of the stored virtual energy is no greater than the amount of energy supplied to the system.

Finally, a dissipative system is considered to be *passive* if a positive-semidefinite storage function $S(x)$ can be found which satisfies Equation (4) when the supply rate $w(t)$ is the product of the input and output of the system:

$$w(t) = u^T(t)y(t) \tag{5}$$

This leads to the *passivity condition* which must be satisfied for a system to be passive:

$$\frac{dS\left(x(t)\right)}{dt} \leq u^T(t)y(t) \tag{6}$$

Or, in shorthand,

$$\dot{S} \leq w$$

Thus, the challenge becomes coming up with differentiable storage functions which follow this rule for systems in question. This process is similar to coming up with Lyapunov functions. Once a system has been shown to be passive, it becomes easier to reason about.

2.2 Stability of Passive Systems

A critically important part of any control problem is determining the stability of the system. This is where the usefulness of passive systems becomes obvious.

THEOREM 1. *If for a system, one can construct a storage function $S(x)$ which is both positive definite and fulfills the passivity condition in Equation (6), then the point $x = 0$ with $u = 0$ is Lyapunov stable.*

Furthermore, if $S(x)$ is radially unbounded, that is, if $\|x\| \to \infty$ implies that $S(x) \to \infty$, then the equilibrium point $x = 0$ is globally stable.

Thus, often by proving that a system is passive, it is easily proved that the system is stable as well.

2.3 Passivity Indices

In practice, not all systems will be passive. It is useful to be able to characterize the degree to which a system is passive or non-passive. This section reviews *passivity indices*, which describe degree of passivity in a system [27].

Considering the same system in Equation (1):

- The system's *input feedforward passivity index* (IFP) is ν if it is dissipative (Equation (4)) with respect to the supply rate
$$w(u(t), y(t)) = u^T(t)y(t) - \nu u^T(t)u(t) \tag{7}$$
for some $\nu \in \mathbb{R}$, denoted as $IFP(\nu)$.
- The system's *output feedback passivity index* (OFP) is ρ if it is dissipative with respect to the supply rate
$$w(u(t), y(t)) = u^T(t)y(t) - \rho y^T(t)y(t) \tag{8}$$
for some $\rho \in \mathbb{R}$, denoted as $OFP(\rho)$.

These ν and ρ are known as the *passivity indices* of the system. If ν or $\rho \geq 0$, then the system is said to have excess passivity, and is said to be *strictly input passive* or *strictly output passive*, respectively. If these indices are negative, then the system lacks passivity. Passivity indices are so named because they provide information on how a system may be made passive by simple connections to other systems, as will be discussed in Section 2.4.

From the given definition, it is obvious that, for instance, if a system is $OFP(2)$, then it is also $OFP(1)$. Generally, the passivity index given is the bound for satisfying Equations (7) and (8). If a system is $OFP(2)$, but is not $OFP(x)$ where $x > 2$, then the system is generally said to be $OFP(2)$, and not $OFP(y)$ where $y < 2$.

2.4 Passivity of Interconnections of Systems

With all these definitions, we now have the mechanisms to examine the way that connecting general systems as building blocks affects the passivity of an overall system.

2.4.1 Parallel and Feedback Connections

An extremely useful and powerful consequence of the passivity of systems is that the parallel and feedback connections of passive systems are also passive. These interconnections are demonstrated in Figures 1 and 2. As feedback control is extremely powerful, knowing that feedback connections are passive is useful in control design.

This statement is a form of the *Passivity Theorem*, proofs and more formal statements of which can be found in [2]

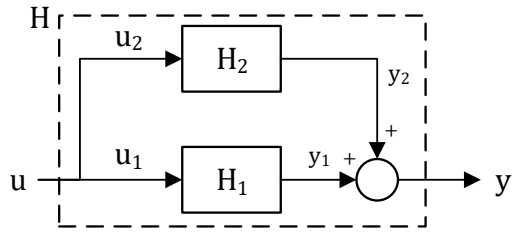

Figure 1: A parallel interconnection of passive systems is passive.

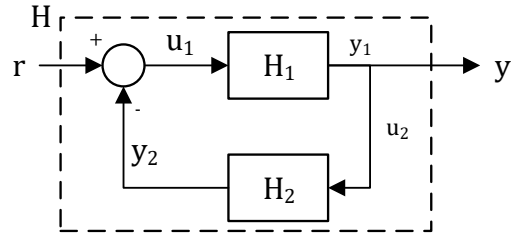

Figure 2: A feedback interconnection of passive systems is passive.

2.4.2 *Passivation of Non-Passive Systems*

Using the inherent passivity of the connections in Section 2.4.1, we can make systems which are not passive act passively when in a parallel or feedback connection with a system which has an excess of passivity, as in Section 2.3.

Input Feedforward Passivation: Consider a general system H (as defined in Equation (1)). If this system is passive with respect to the storage function

$$w(u, y) = u^T y + \nu u^T u \tag{9}$$

where $\nu > 0$, then H is considered to have a lack of input feedforward passivity. Though H is not passive, when connected in a feedforward configuration with a gain νI (as shown in Figure 3), then the whole system $\tilde{H}$ is passive.

Output Feedback Passivation: Similarly, if a system H (Equation (1)) is dissipative with respect to the storage function

$$w(u, y) = u^T y + \rho y^T y \tag{10}$$

where $\rho > 0$, then H is considered to have a lack of output feedback passivity. H can be passified in a feedback configuration with a gain ρI (as shown in Figure 4), and the whole system $\tilde{H}$ is passive.

It is obvious that both Equations (9) and (10) come from Equations (7) and (8). Namely, the passivity indices of systems indicate exactly how much passivity they lack or have

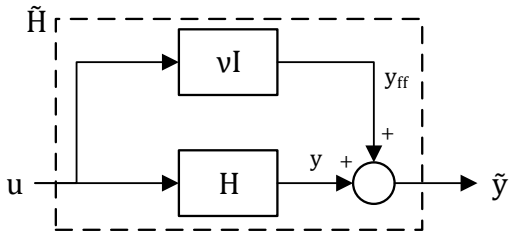

Figure 3: Input feedforward passivation.

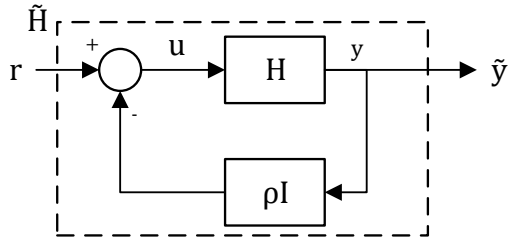

Figure 4: Output feedback passivation.

in excess, and this property defines how another system may be connected to it to compensate.

2.4.3 *Series Connections and the Secant Criterion*

Though equally common, series connections of passive systems are not by definition passive. However, we can get a useful bound on the lack of passivity of such a system using the *secant criterion* [27, 23, 1]. Then, as shown in Section 2.4.2, we can passify these systems using a simple control rule.

Imagine a system of cascaded systems in a negative feedback loop with a control gain K, as shown in Figure 5. We call the number of systems cascaded in this loop n, and for now, consider n to be greater than 2. For now, we will also consider $K = 1$.

If every system in the whole cascaded system is input strictly passive, with $IFP(\nu_i)$ for some $\nu_i \in \mathbb{R}^+$, then each system has a storage function with respect to which it is dissipative, of the form:

$$\dot{S}_i \leq u_i^T y_i - \nu_i u_i^T u_i$$

Then the cascaded system has a storage function which can be considered to be some weighted sum of the storage functions V_i:

$$V = \sum_{i=0}^{n} V_i d_i, \quad d_i > 0 \tag{11}$$

Then, the *secant criterion* states that feedforward cascaded system (without the feedback portion) is, at worst, $OFP(-\rho)$

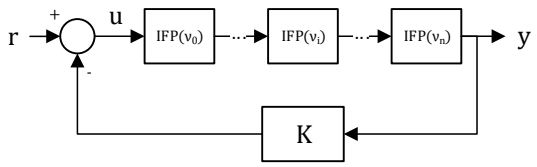

Figure 5: Cascaded systems with output feedback.

where

$$\rho > \frac{\cos\left(\frac{\pi}{n+1}\right)^{n+1}}{\prod_{i=0}^{n} \nu_i} \tag{12}$$

Thus, if put in feedback with a gain ρ, then the overall system will be passive. Similar results and proofs for feedforward passivation and for the case where $n = 2$ can be found in [27].

2.5 Controllability of Passive Systems

Once the passivity of a system is established, then it becomes easier to figure out how to control it. It was established in Section 2.4.1 that the interconnections of passive systems are passive. This, along with passivity indices of these systems, can be used to passify and stabilize both passive and non-passive systems. Because we are talking about non-linear systems which cannot be perfectly modeled, we will chiefly focus on feedback control over feedforward control, as feedback control can account for dynamics in the system.

2.5.1 Proportional Controllers

Passive systems can be easily stabilized with proportional controllers. Imagine that system H (as defined in Equation (1)) is passive with storage function $S(x)$. Then, an output feedback proportional controller using the law $u = -ky$ asymptotically stabilizes the equilibrium of the system ($x = 0$) for any positive k.

PROOF. Because H is passive, it has a storage function $S(x)$ which satisfies

$$\dot{S}(x) \leq u^T y$$

If the control law $u = -ky$ is used, then $\dot{S} \leq -ky^T y < 0$ for all $y \neq 0$. The bounded solution of Equation (1) is within the region of $y = 0$, so $x \to 0$. Thus, the feedback system is passive. □

It is also the case that *all PID controllers* are passive when used in a feedback system, different proofs of which can be found in [2, 13]. Thus, PID controllers can be used to control any passive system, in a passive, stable way.

Importantly, proportional (and PID controllers) can be used to make *non- passive* systems behave in a passive way as well. The output feedback passivity index of the system can be used to determine the amount of proportional control that is required. For instance, if a system is $OFP(-2)$, meaning that is lacks passivity, it can be rendered passive using a gain of 2 in an output feedback loop. Thus, using the passivity index of a system can aid greatly in control design.

2.5.2 Stabilization of Series Interconnections Using Proportional Controllers

By using a proportional controller of a certain gain, it is obvious by combining the information in Sections 2.4.3 and 2.5.1 that a series system which lacks passivity can be made passive using a proportional output feedback controller. However, systems do not necessarily need to be made completely passive to be made stable; instead, information about the lack of passivity can give a bound on the amount of state feedback which is required to simply stabilize the system, which is often the control goal.

Consider the system in Figure 5. We have already established a bound on the output feedback passivity index of such a system in Equation (12). If every block is $IFP(\nu_i)$ where $\nu_i > 0$, then to simply stabilize the feedforward system (again, as opposed to making the entire feedback system stable), then the following criterion must be met, derived from the results of the secant criterion.

$$\frac{1}{K_p \prod_{i=0}^{n} \nu_i} < \frac{1}{\cos\left(\frac{\pi}{n}\right)^n} \tag{13}$$

Here, K_p is the gain of the feedback loop. This criterion gives, in general, a lower bound on the gain of the controller than is required for the full system to be fully passive. Proofs for this criterion, as well as versions for using input feedforward passivity indices, can be found in [27].

2.6 Passivity of Discrete Time Systems

So far, in Section 2, we have dealt with the passivity of systems which are continuous time. Computing and software systems are, by definition, discrete time. Fortunately, many of the powerful results have corresponding discrete time forms which allow for similar analysis to be made [13, 3]. Two things are worth noting. First, that as the sampling period of a discrete time becomes smaller and smaller, approaching continuous time, discrete time becomes a better and better approximation for a continuous time version of the system. Beyond that, [19] introduces the concept of *average passivity*, the idea that a discrete time system may be passive on average when sampling system variables, and averaging them on a certain period. This approach lends itself tidily to software systems, where discrete sampling and averaging is easy.

2.7 Passivity of Delay Systems

In general, delay systems are more difficult to be proved passive, though these conditions exist, especially in linear systems [17, 20]. However, these conditions are beyond the scope of this paper because, in general, discrete-time systems which also have delay cannot be made passive [16, 13]. Because software systems are often non-linear and by definition discrete-time, when delay exists in a software system, it becomes impossible to use passivity theory to reason about the system.

3. AN EXAMPLE SYSTEM

With all of the results from Section 2, we can now use them to reason about a simple software system. This section demonstrates the way that passivity theory can be applied in control design. In the following, we describe a system with multiple components all of which, under certain conditions, can be rendered passive. Then, under these passive conditions, a control design can be achieved which guarantees asymptotic BIBO stability. Finally, we test our control design in a simulator to verify the results from passivity theory.

Our steps are as follows:

1. Determine all system blocks. What are their inputs and outputs?
2. For each system block, determine if the system is passive, and what the passivity indices are. This is done by first determining the passivity of the system by constructing a storage function which satisfies Equation (6). Then, we determine the input passivity index of the system, which satisfies Equation (7).
3. Determine the arrangement of the system blocks, and what the overall passivity of the system is, using the relationships outlined in Section 2.4.
4. Design a controller to compensate for any lack of passivity that the system has.

3.1 System Description and Design

The system we consider and show to be effectively controllable in a passive way is a variation of load balancing. Consider a server farm with N servers, servicing Q different streams of requests. For simplicity's sake, we assume that each of the N servers has the same rate of processing requests, μ. For each of the Q streams q_i, there is a different rate of arrival λ_i. Finally, each q_i has a certain number of servers allocated to it, n_i, where

$$\sum_{i=0}^{Q-1} n_i = N$$

The proprietors of the server farm can then promise total delay ratios to their customers. Imagine a scenario with two kinds of customers: high and low priority customers. Though they cannot guarantee absolutely short delays for their higher priority customers without knowing how many requests they will get in advance (and perhaps having an infinite number of servers), they can promise the total delay experienced by their high priority requests will constitute only a small fraction of the total delay experienced by requests in the system. This idea can be extended to multiple levels of QoS, or quality-of-service. Then, the trick to maintaining this performance is to modify the number of servers per QoS, n_i, such that these delay ratios are met as closely as possible.

As a proxy for delay, we instead use the length of the backlog of each server in a given QoS. This is far easier to measure in a real system than the actual delay, and more instantly gives a picture of the relative amount of work remaining for each QoS level than measuring previous request delays. Importantly, the delay experienced by requests can only be measured after the delay has happened. Delay measurements themselves are inherently delayed! However, measuring the length of queues can be done without any delay, simply by observing the system.

In this paper, we use the variable Q_{i_d} to denote the ratio of queued requests at QoS level i which is desired, and Q_{i_a} to denote the actual measured ratio. Then, the error e being minimized is $e = Q_{i_d} - Q_{i_a}$.

3.1.1 Loop Components

This system as described has several components. Each in turn must be shown to be passive for the system as a whole to be passive, and thus to be able to use non-linear control methods to achieve better performance.

There are four loop components to consider:

1. The component based on the n_i set by the controller. Per QoS, the rate of arrival of requests is λ_i, and cannot be known in advance. However, the effective rate of arrival per server is $\frac{\lambda_i}{n_i}$, and is thus tunable by the controller. This mathematical operation must be represented in the loop as a component.
2. The server itself. Each server has a queue attached to it for requests to sit prior to the server being able to serve them.
3. The performance measurement component of the system. Actual delay measurements are problematic in a system like this, because the delay measurements themselves will be delayed by the amount of delay experienced by a request. Introducing delay into a control loop not only harms control performance of the loop, but in addition, causes the system to not be passive. Instead of actual delay, we instead use the number of queued requests per QoS as a proxy. This component thus computes the actual queue length shares of the system, with each server reporting its queue length. This is used to calculate the control error, which is in turn used by the controller-actuator, closing the loop.
4. The controller component, which determines the change in the number of servers per QoS n_i. This component is essentially the brains of the load balancing portion of the system. Request of QoS q_i come to it, and then they are sent to the appropriate bank of servers.

This system is shown in Figure 6. Specifically, a single control loop for a single QoS level is shown in Figure 7, and a transformed version in Figure 8. This transformation will be discussed in Section 3.1.5.

Because we will use output feedback control, and the main system blocks for each control loop are in a series connection, part of determining the passivity of this system will be determining the input passivity indices for each of these

subsystems to obtain a bound on the passivity of the feedforward system. The next sections show each of these components to be passive, and find these passivity indices. We must set up the system as described in Section 2, and show that the passivity condition in Equation (6) holds, then determine the input passivity indices using the work function in Equation (7).

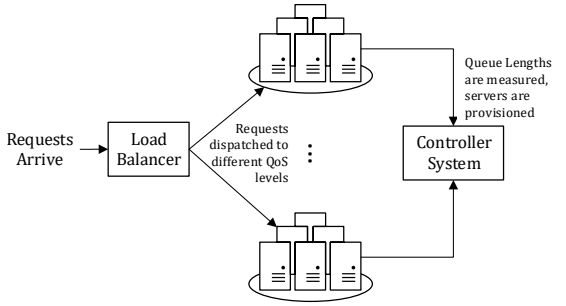

Figure 6: The example system discussed in Section 3.

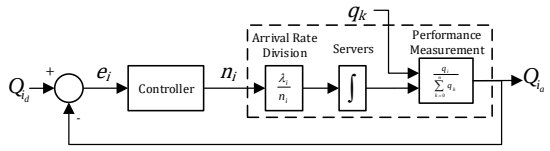

Figure 7: A control loop for a single QoS level for the system in Section 3.

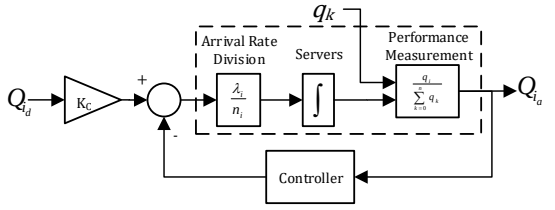

Figure 8: A control loop for a single QoS level for the system in Section 3, with the controller placed in the feedback path instead of the feedforward path.

3.1.2 Passivity of Queues

Here, we formulate queues, which are connected to each of the N servers in the system, to be passive. This formulation of queues relates the request rate and the service rate with the number of requests currently in the queue, or the queue backlog.

Let the request rate be the input: $u(t) = r(t)$, in units of requests per unit time. In our particular system, this is a local request rate for the QoS divided by the number of servers assigned to that QoS. Let the state be the number of requests currently in the queue $q(t)$, and let $q(t)$ also be the output $y(t)$, the variable to be controlled. Finally, we also have the service rate $r_s(t) = \mu$, also in requests per unit time processed. So the system is formulated as follows:

$$\begin{aligned} u(t) &= r(t) \\ x(t) &= q(t) \\ y(t) &= q(t) \\ \dot{x}(t) &= \dot{q}(t) = r(t) - r_s(t) \end{aligned}$$

Note that though, in general, input, output, and state can be vectors, in this component, they are scalars.

PROOF. To show that this system is passive, we construct storage function $S(x)$ such that it fulfills Equation (6). The work function, or supply rate, is

$$w(t) = u(t)y(t) = r(t)q(t)\,\frac{\text{requests}^2}{\text{time}}$$

The only state in this system is $q(t)$, so S must be of the form $S(q(t))$. Try

$$S(x) = \frac{1}{2}\kappa_q q(t)^2\,\frac{\text{requests}^2}{\text{time}}$$

Here, κ_q is an adjustable constant with units $\frac{1}{\text{time}}$. This storage function is positive definite; the storage function is equal to zero if and only if the length of the queue is zero.

So, checking Equation (6):

$$\begin{aligned} w(t) = q(t)r(t) &\geq \frac{dS(x)}{dt} \\ &\geq \frac{d}{dt}\left(\frac{1}{2}\kappa_q q(t)^2\right) \\ &\geq \kappa_q q(t)\dot{q}(t) \\ q(t)r(t) &\geq \kappa_q q(t)\left(r(t) - r_s(t)\right) \\ r(t) &\geq \kappa_q\left(r(t) - r_s(t)\right) \end{aligned}$$

So, so long as $r(t) \leq r(t) - r_s(t)$, namely, that the service rate of the queue is positive (which we can assume with confidence), and that κ_q is set appropriately, the passivity equation holds. Furthermore because the storage function is positive definite, the system is stable around the equilibrium point $q(t) = 0$. □

Each server in this system can be considered to be a queue with an average service rate. The controller does not have to do any special work to ensure that this subsystem is passive.

Next, we determine the input passivity index of the queue. Use the work function from Equation (7):

$$\begin{aligned} w &= u^T y - \nu u^T u \\ &= rq - \nu r^2 \end{aligned}$$

The storage function we use can be the same from the proof above:

$$S(x) = \frac{1}{2}q(t)^2$$
$$\frac{dS(x)}{dt} = q(t)\dot{q}(t)$$

Then, plug into Equation (6):

$$\begin{aligned} w &\geq \dot{S} \\ rq - \nu r^2 &\geq q\dot{q} \\ rq - \nu r^2 &\geq q\left(r - r_s\right) \\ qr - \nu r^2 &\geq qr - qr_s \\ \nu r^2 &\leq qr_s \\ \nu &\leq \frac{qr_s}{r^2} \end{aligned}$$

Thus, this formulation of queues is $IFP(\frac{qr_s}{r^2})$. This quantity is always positive by definition, and is greater than 0 for all $q > 0, r > 0$. This makes intuitive sense as well; if there are no requests arriving at the queue, then $\nu = \infty$, meaning that the system is infinitely passive.

3.1.3 Passivity of Arrival Rate Division

In this section, we cover the passivity of the arrival rate division that occurs when the number of servers allocated to a single QoS changes. Each QoS level has an arrival rate $\lambda_i(t)$. Each QoS level has $n_i(t)$ allocated to it. Therefore, each individual server in this QoS level experiences an individual arrival rate of $\frac{\lambda_i(t)}{n_i(t)}$. The system can be formulated as follows, using the shorthand $\lambda_i(t) = \lambda$ and $n_i(t) = n$:

$$\begin{aligned} u(t) &= \begin{bmatrix} n & \lambda \end{bmatrix}^T \\ x(t) &= \frac{\lambda}{n}\ \frac{\text{requests}}{\text{servers} \cdot \text{time}} \\ y(t) &= \frac{\lambda}{n} \\ \dot{x}(t) &= \frac{\dot{\lambda}}{n} - \lambda\frac{\dot{n}}{n^2}\ \frac{\text{requests}}{\text{servers} \cdot \text{time}^2} \end{aligned}$$

Proof. To show that this system is passive, we again construct storage function $S(x)$ such that it fulfills Equation (6). The supply rate is

$$\begin{aligned} w(t) &= u^T(t)y(t) \\ &= \begin{bmatrix} \lambda & n \end{bmatrix}\frac{\lambda}{n} \\ &= \frac{\lambda^2}{n} + \lambda \\ &= \frac{\lambda}{n}\lambda + \frac{\lambda}{n}n \\ &= x(t)(\lambda + n) \end{aligned}$$

Let the storage function and its derivative be

$$\begin{aligned} S(x) &= \frac{1}{2}x^2 = \frac{1}{2}\frac{\lambda^2}{n^2} \\ \dot{S}(x) &= \dot{x}x \end{aligned}$$

This storage function is positive definite, since it cannot be zero unless all the state variables are zero, and because the state variables are squared.

So, to verify Equation (6):

$$\begin{aligned} \dot{S}(x) &\leq w \\ \dot{x}x &\leq x(\lambda + n) \\ \dot{x} &\leq \lambda + n \\ \frac{\dot{\lambda}}{n} - \lambda\frac{\dot{n}}{n^2} &\leq \lambda + n \\ \frac{\lambda}{n}\left(\frac{\dot{\lambda}}{\lambda} - \frac{\dot{n}}{n}\right) &\leq \lambda + n \\ \frac{\dot{\lambda}}{\lambda} - \frac{\dot{n}}{n} &\leq n + \frac{n^2}{\lambda} \\ \frac{\dot{\lambda}}{\lambda} &\leq \frac{\dot{n}}{n} + n + \frac{n^2}{\lambda} \end{aligned}$$

In this expression, λ and $\dot{\lambda}$ are measured system constants. In a real system, n is the number of servers being assigned to the QoS level, and $\dot{n}$ is the change from the last time the controller was run to this time. So, we can replace the continuous time derivative with a discrete approximation using the finite difference method:

$$\dot{n} = \frac{n - n_{old}}{\Delta t}$$

Therefore, this expression is a function of measured constants and n, which can be adjusted by the controller in such a way that this system is passive. □

This equation for n is of the form:

$$0 = \frac{1}{n} + 1 + n + n^2$$

These equations have three solutions for n, but only one of them is positive and real, which n, as a physical number of servers, must be. Numerically, the solution to the passivity condition above gives a lower bound to the number of servers assigned to a single QoS level. In practice, this lower bound turns out to be low enough that it is easy to meet; in fact, in all of the reasonable cases simulated, this lower bound was 1 server.

Next, we determine the input passivity index of this system, using the work function from Equation (7):

$$\begin{aligned} w &= u^T y - \nu u^T u \\ &= \begin{bmatrix} \lambda & n \end{bmatrix}\frac{\lambda}{n} - \nu\begin{bmatrix} \lambda & n \end{bmatrix}\begin{bmatrix} \lambda \\ n \end{bmatrix} \\ &= \frac{\lambda}{n}\left(\lambda + n\right) - \nu\left(\lambda^2 + n^2\right) \end{aligned}$$

We use the same storage function from the passivity proof:

$$\begin{aligned} \dot{S}(x) &= \dot{x}x \\ &= \left(\frac{\dot{\lambda}}{n} - \lambda\frac{\dot{n}}{n^2}\right)\frac{\lambda}{n} \end{aligned}$$

Finally, plug into Equation (6):

$$\begin{aligned}
\dot{S} &\leq w \\
\frac{\lambda}{n}\left(\frac{\dot{\lambda}}{n} - \lambda\frac{\dot{n}}{n^2}\right) &\leq \frac{\lambda}{n}(\lambda + n) - \nu\left(\lambda^2 + n^2\right) \\
\frac{\lambda}{n}\left(\frac{\dot{\lambda}}{n} - \lambda\frac{\dot{n}}{n^2} - \lambda - n\right) &\leq -\nu\left(\lambda^2 + n^2\right) \\
\nu &\leq -\frac{\frac{\lambda}{n}\left(\frac{\dot{\lambda}}{n} - \lambda\frac{\dot{n}}{n^2} - \lambda - n\right)}{\lambda^2 + n^2}
\end{aligned}$$

This expression, though inelegant, gives us a calculable bound on ν for the arrival rate division system. All of these variables are measured system parameters, including the derivatives, which can be accounted for using finite differences, as above. Thus, depending on n, the system can be more or less passive.

3.1.4 Passivity of Performance Measurement

In [15, 14], the authors present a method for controlling relative delay ratios in web servers. Because of the nondeterministic nature of arriving requests, the delay promises made can only be relative, but this is a reasonable proxy for actual delay in normal operating conditions. We take this abstraction a step further by using the number of queued requests in each QoS level as a proxy for the total queuing delay, which we assume dominates the total delay a request experiences in this system. The advantage to this is that the length of the queues in the system can be measured directly at an instant in time, whereas delay can only be measured after the delay has occurred. This tightens the control loop and makes the passivity of the system easier to reason about, as discussed.

Let the input to this block be the number of requests queued in each QoS level. This can be obtained by summing the length of the queues at each individual server. Let q_i represent this sum for a QoS level i at which this particular block is operating, and q_i can also be taken to be the state of the block. Then, the output of the block is the share of the total number of queued requests that QoS level i bears. In more formal language,

$$\begin{aligned}
u(t) &= \begin{bmatrix} q_0 & q_1 & \cdots & q_i & \cdots & q_n \end{bmatrix}^T \\
x(t) &= q_i \\
y(t) &= \frac{q_i}{\sum_{k=0}^{n} q_k}
\end{aligned}$$

Note that all q_i's are a function of time, so q_i is a shorthand for $q_i(t)$.

Proof. To show that this system is passive, we again construct a storage function $S(x)$ such that it fulfills Equation (6). The supply rate is

$$\begin{aligned}
w(t) &= u^T(t)y(t) \\
&= \begin{bmatrix} q_0 & q_1 & \cdots & q_i & \cdots & q_n \end{bmatrix} \frac{q_i}{\sum_{k=0}^{n} q_k} \\
&= \frac{q_i q_0}{\sum_{k=0}^{n} q_k} + \cdots \frac{q_i^2}{\sum_{k=0}^{n} q_k} + \cdots \frac{q_n q_i}{\sum_{k=0}^{n} q_k} \\
&= q_i\left(\frac{q_0 + \cdots + q_i + \cdots + q_n}{\sum_{k=0}^{n} q_k}\right) \\
&= q_i \frac{\sum_{k=0}^{n} q_k}{\sum_{k=0}^{n} q_k} \\
&= q_i
\end{aligned}$$

Take κ to be a constant in units of requests per time, and set it equal to the maximum number of requests that this QoS level will experience at any given time. This constant can be arbitrarily high, but is not infinite, as we can reasonably assume that an unbounded number of requests will not arrive at our system in a bounded amount of time. Then, a possible storage function $S(x)$ is

$$S(x) = \frac{1}{2\kappa} q_i^2 \text{ time} \cdot \text{requests}$$

Moreover, if κ is positive, then this function is positive definite.

Then, to verify Equation (6):

$$\begin{aligned}
w(t) = q_i &\geq \frac{dS(x)}{dt} \\
&\geq \frac{d}{dx}\left(\frac{1}{2\kappa} q_i^2\right) \\
&\geq \frac{1}{\kappa} \dot{q}_i q_i \text{ requests}
\end{aligned}$$

Because $\dot{q}_i$ is at most κ, this expression is at most q_i, so the passivity condition holds. □

Next, we find the input passivity index of the performance measurement system, as done in the previous sections. First, calculate the work function in Equation (7):

$$\begin{aligned}
w &= u^T y - \nu u^T u \\
&= q_i - \nu \sum_{k=0}^{n} q_k^2
\end{aligned}$$

Use the same storage function, with the same derivative, as above:

$$\dot{S} = \frac{1}{\kappa} \dot{q}_i q_i$$

Then, plug into Equation (6):

$$\begin{aligned}
\dot{S} &\leq w \\
\frac{1}{\kappa} \dot{q}_i q_i &\leq q_i - \nu \sum_{k=0}^{n} q_k^2 \\
\nu &\leq \frac{q_i - \frac{1}{\kappa} \dot{q}_i q_i}{\sum_{k=0}^{n} q_k^2} \\
\nu &\leq \frac{q_i}{\sum_{k=0}^{n} q_k^2}\left(1 - \frac{1}{\kappa} \dot{q}_i\right)
\end{aligned}$$

If we make κ arbitrarily high, then we can eliminate the second term and get a bound on the passivity index of this system.

$$\nu \leq \frac{q_i}{\sum_{k=0}^{n} q_k^2}$$

We are permitted to do this because no matter how large κ is, the storage function is still positive definite. This convenience of having a manipulable storage function, which is not necessarily related to a true physical quantity, is common in software systems. Though ν might be small, it is always greater than zero, meaning that this system has a small excess of passivity. Furthermore, it is calculable based on system parameters.

3.1.5 Passivity of the Controller

The most important component of the loop, and the one over which we have the most control in its design, is the controller of the system. In this design, every QoS level has its own controller in the loop, and these controllers coordinate to determine the number of servers which should be changed to handle requests at each QoS level.

As discussed in Section 2.5.1, all proportional controllers and PID controllers are passive when used in a feedback loop. This system can easily be transformed from the system where the controller is in the feedforward path (as in Figure 7) to where the controller is in the feedback loop (as in Figure 8), provided the controller is LTI. [18] Thus, though a proportional controller might not be passive from its input to output on its own, using a proportional controller in the feedback loop will preserve the passivity of the system as a whole.

3.1.6 Putting It All Together

The three components of the system are passive, but their series connection may or may not be passive. We use the secant criterion from Section 2.4.3 to determine the upper bound on the passivity of the feedforward system. The expressions for the input passivity indices of each system are:

$$\nu_{\text{Arrival Rate Division}} = \nu_a = -\frac{\frac{\lambda}{n}\left(\frac{\dot{\lambda}}{n} - \lambda\frac{\dot{n}}{n^2} - \lambda - n\right)}{\lambda^2 + n^2}$$

$$\nu_{\text{Server Queue}} = \nu_q = \frac{q r_s}{r^2}$$

$$\nu_{\text{Performance Measurement}} = \nu_p = \frac{q_i}{\sum_{k=0}^{n} q_k^2}$$

There are three components, so $n = 3$ in the secant criterion expression in Equation (12). Then, the output feedback passivity index of the whole feedforward system is bounded by

$$\rho_{\text{Feedforward System}} = \rho_s > -\frac{\cos\left(\frac{\pi}{4}\right)^4}{\nu_a\nu_q\nu_p} = -\frac{1}{4\nu_a\nu_q\nu_p} \qquad (14)$$

provided that $\nu_{a,q,p} > 0$. Note that this bound shows that, at worst, the system is not passive! To guarantee passivity of the overall system, we can make up for the lack of passivity in the feedforward system by increasing the controller gain in the feedback system, as discussed in Section 2.4.2.

For example, if the following system parameters are measured in a single control cycle:

$$\nu_a = \frac{1}{4} \quad \nu_q = \nu_p = 1$$

then,

$$\rho_s = -\frac{1}{4\frac{1}{4}} = -1$$

So, the gain of the controller must be at least 1 to make the entire system passive.

An even laxer bound on the gain of the controller comes from the expression in Equation (13), which stabilizes without making the whole system passive. We can use the same passivity indices to compute this:

$$\frac{1}{K_p\nu_a\nu_q\nu_p} < \frac{1}{\cos\left(\frac{\pi}{3}\right)^3}$$

$$\frac{8}{\nu_a\nu_q\nu_p} < K_p \qquad (15)$$

Therefore, without linearizing, or tuning, we have a calculable bound for the passivity of this system, which in turn can inform our control of this system. In the next section, we examine the behavior of such a system with a software simulator, under a few different conditions.

3.2 The Simulator

The software simulator for this system was built using Python 3.5.1 [21], NumPy to handle array and numerical operations [25], and SimPy, an event-based simulation library [24]. SimPy provided event scheduling like running the controller at fixed intervals, simulating event arrivals and services, etc.

3.2.1 Simulator Parameters

The simulator was built to be configurable. However, the results presented here represent the following set simulator parameters:

- $N = 1000$: The simulator was run with 1000 servers.
- $T = 100$: The controller period was 100 seconds. Different system parameters required for calculating passivity indices are measured over the control period and averaged (as discussed in Section 2.6).
- Total Time = 5000: The simulator was run for 5000 seconds.

3.2.2 Arrival Profiles

For the purposes of simulation, simulated events arrive in an exponential distribution with a certain mean λ, and then take a certain time to process, which is also distributed exponentially. For most experiments, the simulated server farm was run at 80% to 90% of the theoretical capacity of the farm. Requests took 10 seconds to process, and depending on the service quality, and time in the simulation, arrived at different rates. Four different arrival profiles with 2 QoS levels, and one with 3 QoS levels, were tested. Example traces measuring the arrival rates for these five different traffic arrival profiles are in Figures 9 to 13.

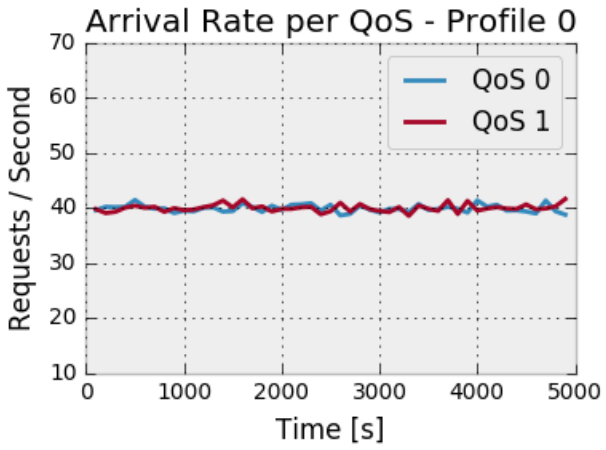

Figure 9: The traffic arrival profile 0.

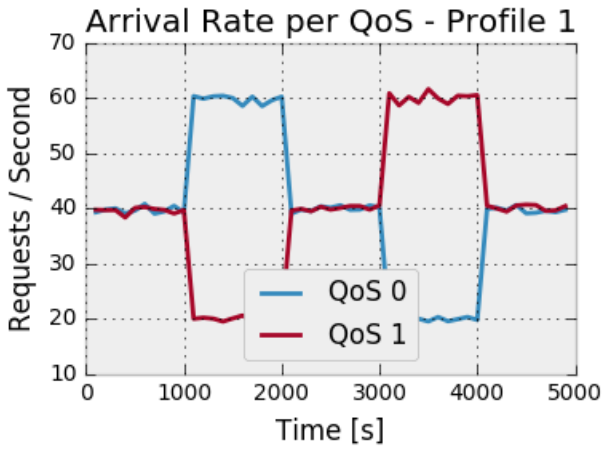

Figure 10: The traffic arrival profile 1.

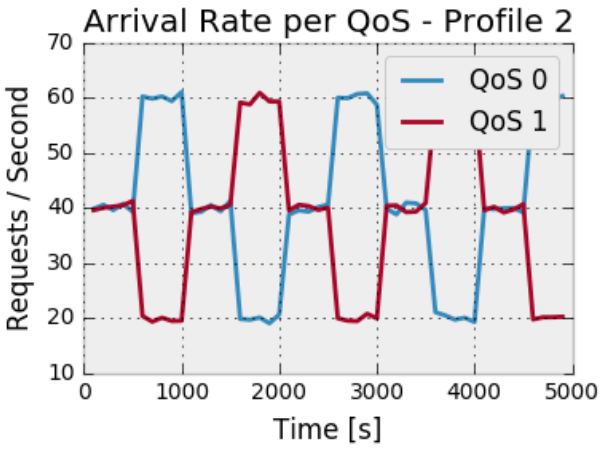

Figure 11: The traffic arrival profile 2.

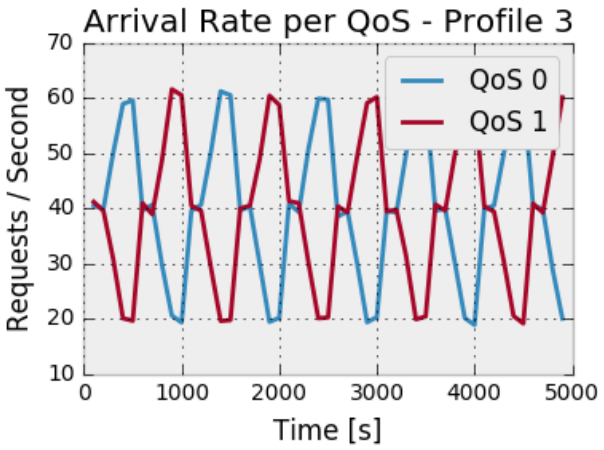

Figure 12: The traffic arrival profile 3.

3.2.3 Simulated Controllers

Three different proportional controllers were simulated:

1. A static proportional controller. This is labelled 'p' in the results.
2. A dynamic proportional controller which passified the entire system. It uses whatever proportional gain was required to make the entire system passive using the expression in Equation (14), as discussed in Section 3.1.6.
3. A dynamic proportional controller which only stabilized the system. This controller used the gain determined by using the stabilizing expression in Equation (15), as also discussed in Section 3.1.6.

3.3 Simulation Results

In general, the controllers informed by passivity performed well, with a few caveats. Summaries of average errors are in Figure 14. Sample queue shares by time charts for all three controllers, for Arrival profile 0, are in Figures 15 to 17. Both passivity theory based controllers performed far better than a simple static proportional controller, especially at a steady state. However, in achieving such low steady state error, these controllers were extremely aggressive, and allocated as many servers as possible to one QoS, or the other (the simulator ensured each QoS had at least one server). Due to this, the queue share measured was very desirable, but the queues of a few servers grew very large. Without job rebalancing modeled in the passivity of the system, the controllers could not account for this properly.

4. EXAMINING REAL SYSTEMS

With an example system tried and tested, now we turn to analysis of more complex, real software systems. The same concepts can be applied as in Section 3, with varying degrees of success. In particular, we will look at two different published software systems: Proteus and Pyro.

4.1 Proteus

Proteus [12] is a software system which builds on memcached [5], modifying it to increase performance of a cache cluster

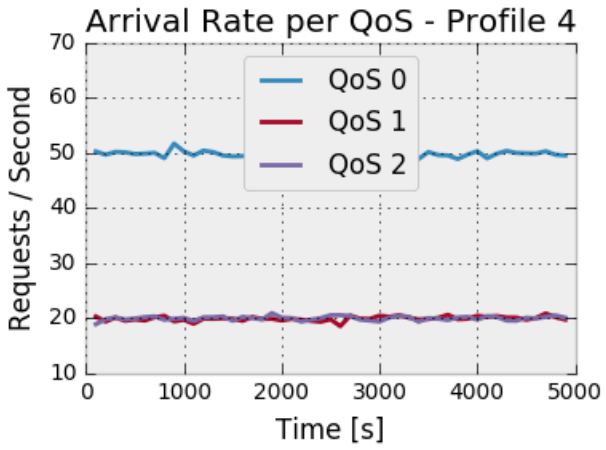

Figure 13: The traffic arrival profile 4.

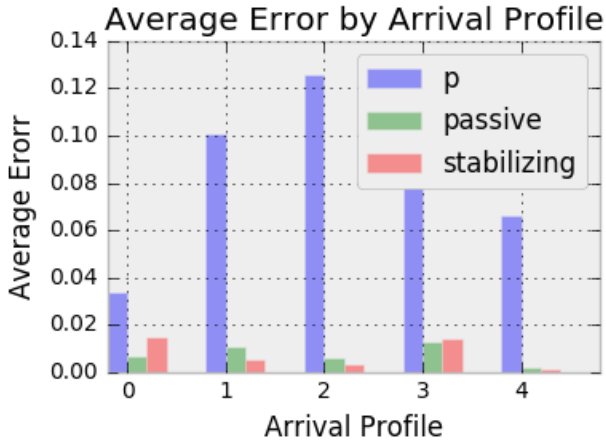

Figure 14: Average errors for each tested controller, by traffic arrival profile.

by improving the provisioning of cluster nodes, specifically in order to save energy by turning underutilized cache nodes off without impacting performance.

4.1.1 *Proteus System Design*

There are a few different system components in Proteus, based on the goals of the system. Proteus aims to:

- Balance the load on each cache node, even when the amount of work is changing, and the number of nodes is similarly changing.
- Minimize the movement of data between cache nodes during rebalancing.
- Eliminate delay spikes during rebalancing.

The end result is that Proteus is an actuator which, when told, will create smooth transitions between different cache provisioning systems. Thus, Proteus can be used to turn

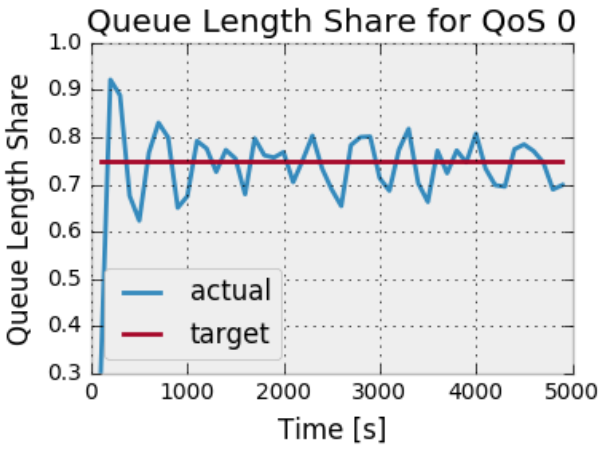

Figure 15: Queue share over time for QoS 0, Arrival Profile 0, for a simple proportional controller.

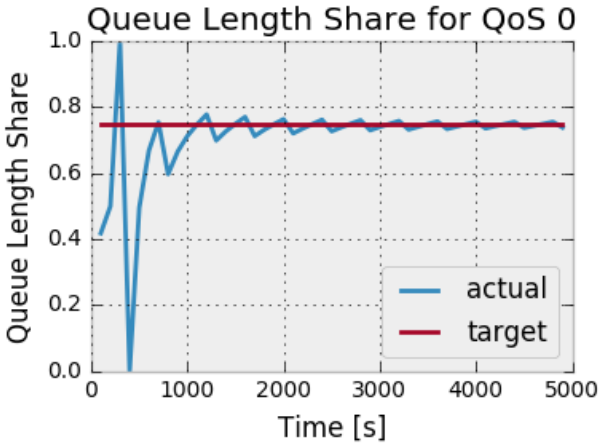

Figure 16: Queue share over time for QoS 0, Arrival Profile 0, for a passive controller.

servers on and off when loads are high and low, in order to save energy in a data center.

In order to provide these smooth transitions, Proteus employs a Bloom filter [4] to count memcached data keys, which allows for Proteus to know what data is "hot", and should be moved to a different cache node if the cache node it is on is to be turned off.

Proteus itself can be used with any provisioning system (as noted in the Proteus paper [12]), which means that any sort of control loop can be used, with Proteus as the cache cluster actuator. In the experiments reported in the Proteus paper, delay was the measured and controlled variable. When average delay reached a certain threshold over the desired delay per request, more cache nodes were turned on.

4.1.2 *Proteus Passivity Analysis*

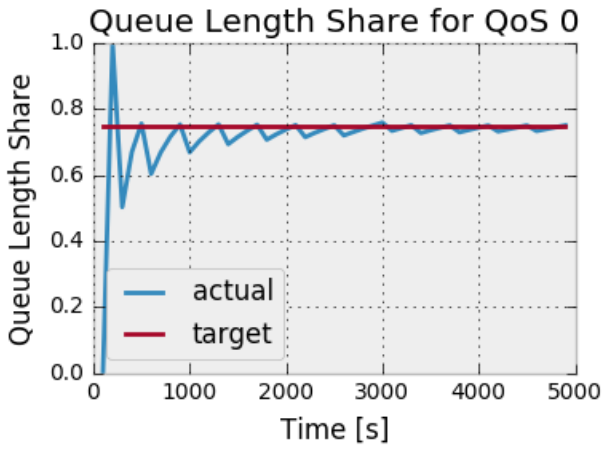

Figure 17: Queue share over time for QoS 0, Arrival Profile 0, for a stabilizing controller.

For the purposes of passivity analysis of Proteus, the system can be simplified greatly. Notably, the way that the control loop is designed in the Proteus paper presents two major problems for passivity analysis:

1. *Delay measurements are inherently delayed.* As discussed in Section 3.1.4, the reason queue length was measured in the example system is that by the time the delay experienced by a request is determined, that request has left the system and the measurement may no longer reflect what is happening in the system. As such, delay measurements are not passive. The reason proportional controllers work is that the direction of change is always guaranteed to bring the system closer to the set point. But when the measurement is delayed, then the direction of change is also delayed, and the controller is no longer passive.
2. *The Bloom filter, which smooths transitions, also introduces delay.* Even if the measurements of delay were not a problem, the main contribution of the Proteus paper, the smoothing of the provisioning, causes the same kind of direction of change delay. Though in practice it works well, in theory, the delay in the actuation caused by the Bloom filter is not passive.

Thus, passivity analysis of Proteus is untenable, unless large changes are made to the main thrust of the system.

4.2 Pyro

Pyro [11] is a geo-spatial data-store designed for optimizing the speed and efficiency of geometrically and temporally collocated queries, which are common to many kinds of data. It was made by making modifications to Apache Hadoop [26], HBase [6], and HDFS [22], all of which allow Pyro to understand geometry associated with data.

4.2.1 Pyro System Design

Though there are many components to the design of Pyro, the ones which are relevant to the design of a control loop for it are those related to deciding when to split and join geo-spatial regions in the data. For instance, if a race was being run through a city, and the runners were at different places in different times, then the hot spot of runner location data would move through space over time, and ostensibly, be handled by different HRegion servers. It would be inefficient to evenly space the HRegion servers simply by geometry; instead, Pyro can split and join regions based on the amount of data stored there, and how many requests for said data are present.

Thus, the control loops must tell each region of data whether it should split, join or do neither. This can be done by simple thresholding, or by measuring the share of work each region is doing, similar to the example system in Section 3. Thus, we can do passivity analysis of control loops which we can set up in a way that the original Pyro paper suggests [11].

4.2.2 Pyro Passivity Analysis

A key difference between Pyro and Proteus is that as an actuator, Pyro does not have intentional delay in its split (or join) operations. Thus, the delay which ruined the passivity analysis from Section 4.1 on Proteus will not plague Pyro in the same way. In fact, Pyro is a perfect example of a system which follows the same pattern as the example system in Section 3, with a few key differences:

1. Instead of measuring queue length, it would be more useful for Pyro to measure the share of requests being serviced in a given control period, and ensure that that share is close to even. Imagine that there are N regions. Each region receives requests at a rate of λ_i in a control period. Then, a single region's request burden b_i would be:

$$b_i = \frac{\lambda_i}{\sum_{k=0}^{N-1} \lambda_k}$$

   Then, the error of the control loop for that region would be:

$$e = b_i - \frac{1}{N}$$

   This performance measurement block can be shown to be passive in the exact same way as the performance measurement block in Section 3.1.4.
2. The number of servers N is not necessarily fixed. It is possible in a Pyro deployment scenario that extra servers are provisioned only when necessary, to save energy. Though this changes the number of control loops and inputs to blocks after any given control decision, this does not change the passivity analysis from Section 3.

Therefore, a control loop controlling a Pyro system can be informed by passivity theory, in a very similar way to the results in Section 3.

5. RELATED WORK

The notion of passivity has been around in electrical and mechanical systems since the 1970s [2], but only recently has the notion of using passivity theory to analyze software-related systems become a topic of research. Recently, in the

digital realm, passivity theory has been used to design controllers for robots [7], examine the propagation of viruses [9], and examine the passivity properties of neural networks [10]. However, there is very little research being done on this topic. Moreover, the vast majority of software developers do not have basic control theory, let alone passivity theory, in their toolboxes. We hope that this project will help software developers understand how to use passivity theory in their work to develop more robust and better performing software.

6. CONCLUSIONS

We have presented a framework for analyzing software systems using passivity theory, which promises to be useful for software control design in many scenarios. In the future, software engineers should be able to use passivity theory to design and analyze their projects, recognize places where control theory can be used to improve performance, and use the properties of passive systems to aid in control design.

Acknowledgments

This material is based in part upon work supported by the National Science Foundation under Grant Number CNS-1320209. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] M. Arcak and E. D. Sontag. Diagonal stability of a class of cyclic systems and its connection with the secant criterion. *Automatica*, 42(9):1531–1537, 2006.

[2] J. Bao and P. Lee. *Process Control: The Passive Systems Approach*. Advances in Industrial Control. Springer London, 2007.

[3] I. Barkana, H. Weiss, and I. Rusnak. On implementing passivity in discrete linear systems. In *Control and Automation (ICCA), 2011 9th IEEE International Conference on*, pages 1009–1014. IEEE, 2011.

[4] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.

[5] B. Fitzpatrick. Distributed caching with memcached. *Linux J.*, 2004(124):5–, Aug. 2004.

[6] L. George. *HBase: The Definitive Guide*. O'Reilly Media, Inc., 2011.

[7] T. Hatanaka, N. Chopra, and M. W. Spong. Passivity-based control of robots: Historical perspective and contemporary issues. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 2450–2452. IEEE, 2015.

[8] I. D. Landau, R. Lozano, M. M'Saad, and A. Karimi. *Adaptive Control*, volume 51. Springer Berlin, 1998.

[9] P. Lee, A. Clark, L. Bushnell, and R. Poovendran. Passivity framework for composition and mitigation of multi-virus propagation in networked systems. In *2015 American Control Conference (ACC)*, pages 2453–2460. IEEE, 2015.

[10] H. Li, H. Gao, and P. Shi. New passivity analysis for neural networks with discrete and distributed delays. *IEEE Transactions on Neural Networks*, 21(11):1842–1847, 2010.

[11] S. Li, S. Hu, R. Ganti, M. Srivatsa, and T. Abdelzaher. Pyro: a spatial-temporal big-data storage system. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 97–109, 2015.

[12] S. Li, S. Wang, F. Yang, S. Hu, F. Saremi, and T. Abdelzaher. Proteus: Power proportional memory cache cluster in data centers. In *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, pages 73–82. IEEE, 2013.

[13] R. Lozano, B. Maschke, B. Brogliato, and O. Egeland. *Dissipative Systems Analysis and Control: Theory and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2000.

[14] C. Lu, T. Abdelzaher, J. A. Stankovic, and S. H. Son. A feedback control approach for guaranteeing relative delays in web servers. In *Real-Time Technology and Applications Symposium, 2001. Proceedings. Seventh IEEE*, pages 51–62. IEEE, 2001.

[15] Y. Lu, T. Abdelzaher, C. Lu, L. Sha, and X. Liu. Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers. In *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*, pages 208–217. IEEE, 2003.

[16] C. Ma and M. Vidyasagar. Nonpassivity of linear discrete-time systems. *Systems & Control Letters*, 7(1):51–53, 1986.

[17] M. S. Mahmoud and A. Ismail. Passivity and passification of time-delay systems. *Journal of Mathematical Analysis and Applications*, 292(1):247–258, 2004.

[18] C. Mei. On teaching the simplification of block diagrams. *International Journal of Engineering Education*, pages 697–703, 2002.

[19] S. Monaco, D. Normand-Cyrot, and F. Tiefensee. From passivity under sampling to a new discrete-time passivity concept. In *2008 47th IEEE Conference on Decision and Control*.

[20] S.-I. Niculescu and R. Lozano. On the passivity of linear delay systems. *Automatic Control, IEEE Transactions on*, 46(3):460–464, 2001.

[21] G. Rossum. Python reference manual. Technical report, Amsterdam, The Netherlands, The Netherlands, 1995.

[22] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10. IEEE, 2010.

[23] E. D. Sontag. Passivity gains and the "secant condition" for stability. *Systems & Control Letters*, 55(3):177–183, 2006.

[24] Team SimPy. Welcome to SimPy, 2016.

[25] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science and Engg.*, 13(2):22–30, Mar. 2011.

[26] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[27] H. Yu and P. J. Antsaklis. Passivity of cascaded systems based on the analysis of passivity indices. *ISIS*, 9:008, 2009.