Predicting Performance on a Loosely Controlled Event System

S. Rooney and L. Garcés-Erice
IBM Research, Zurich Laboratory 8803 Rüschlikon, Switzerland
E-mail: \{sro, lga\}@zurich.ibm.com

Abstract

We contend that event systems will become the kernel of distributed middleware for multicore systems due to their asynchronous nature. In a concurrent system, the fewer points of synchronization between independent activities, the better the achievable parallelization. As physical devices are integrated into distributed middleware the timeliness of the execution of functions becomes increasingly important. Precise guarantees require complete control of all the elements which influence the execution. However, the purpose of middleware is to abstract from the exact nature of the underlying platform. Here we outline an approach that allocates processor shares to event queues, thus controlling the rate of events handled at any given queue. We show how this in turn allows us to predict the likely throughput through a queue. This prediction is achieved through model fitting and calibration.

1. Introduction

Queued event systems detach the production of an event from its handling. Structuring applications through the use of such abstractions allows a looser form of coupling than via function calls. When the queues themselves are lock-free then there is no synchronization at all between event producers and handlers. In a multiprocessor environment this helps in proving the correctness of the program and allows better parallelization. Moreover, as the operational semantics of placing an event in a remote queue are the same as that of placing it in a local one, they do not suffer the same impedance as remote procedure calls where the distributed system attempts to give the illusion that a remote network invocation has the same semantic as pushing a new frame onto the stack [1].

For these reasons we expect a queued event approach to become the unifying abstraction for communication within distributed middleware, whether the communication is intra or inter machine. An example application is real-time analytics, where multiple data sources are analyzed in real-time for different underlying signals, e.g. in order to detect fraud. The SEDA system [2] is an example of a distributed middleware architecture based on staged event queues.

As event production and handling are asynchronous the event producer cannot know when the event will actually be handled. In an environment in which the timeliness of the execution of operation is important this is prohibitive. Real-time systems have traditionally been conceived of as a set of tasks coordinated by a scheduler according to some scheduling discipline, for example [3], in which the scheduler divides the CPU cycles amongst the tasks. We propose a model in which the shares are allocated to typed event queues rather than to tasks. We share resources among the event queues by use of a variant of weighted fair queuing (WFQ) [4]. Each queue has a specific handling function associated with it. We denote the combination of the typed event queue and the associated handler function a topic.

First we give a formal description of the throughput guarantee that the scheduler gives, this takes into account the work conserving nature of the scheduler such that a topic is not given a guarantee in isolation but uses information about the setting of the entire system. In particular, it requires knowledge of the maximum sustainable throughput $R_{\text{max}}$ at that setting.

We show that our implementation of the scheduler is faithful to the model by comparing the predicted and actual measured throughput for topics at settings with known $R_{\text{max}}$. Then we show how we estimate the $R_{\text{max}}$ of unknown settings from those of known ones.

2. Description of the Event Scheduler

2.1. Overview

As [2] states, the key property of scalability under load is graceful degradation, i.e. the penalty incurred is predictable and controllable. Previous work has described a topic scheduler for use on a multicore system [5]. This scheduler always chooses to schedule the
topic with the most number of credits that has a non-empty queue. The credits of all topics are reset when no non-empty topic has any credit, i.e. the scheduler is work conserving. The total credits allocated to a topic is a fraction of a time period $T$. Each time a topic is scheduled the scheduler measures how long the associated handling functions runs before it completes. This amount is deducted from the total number of credits for that topic. If some of the topics with allocated credits have no work to do in a period, the duration of that period may be less than $T$. Such a scheduler can be efficiently implemented on a multiprocessor system through a set of threads that coordinate through a lock-free concurrent priority queue. A description of the Java implementation and the formal properties of the data structure are given in [5]. This implementation currently supports a distributed publish/subscribe system. We use this particular incarnation of the scheduler to test the models derived here. In the publish/subscribe system the event queues correspond to message topics and the events are the arrival of in-going or out-going messages. Out-going messages are transmitted over the network to all subscribers, while in-coming messages are dispatched to the handling application. In our description of the performance of the scheduler we treat “event” and “message” as interchangeable.

Note that the topic scheduler is independent of the OS thread scheduler and the share that it guarantees only refers to the time that the topic scheduler itself runs. We shall assume in our initial description that everything on the system is driven by the scheduling threads: this ignores additional activities such as I/O and garbage collecting. These are implicitly accounted for within the calibration we describe in Section 4.

The actual time that the scheduler takes to service a single event in a given type of event queue is measured by the scheduler, so the average handling time per event is calculable. It is tempting then to conclude that it is sufficient to allocate shares subject to the constraint that the fraction of $T$ allocated to a topic is less than the product of expected arrivals and average event handling time, and that the sum of the products for all topics is less than $T$. This is overly simplistic for two reasons: (1) it is too conservative; as the scheduler is work conserving, events in queues with no credit can still be handled if a topic with credit is idle; (2) the total number of events the scheduler can handle during a time period $T$ changes with the number of topics, their share allocation and their offered load in ways which are difficult to predict without measurement.

We address the second point in Section 4; we now give a formal model which takes into account the work conserving nature of the scheduler.

### 2.2. Formal Model

Multiple scheduling threads can schedule distinct topics in parallel, however a given topic cannot be scheduled simultaneously by two threads. This constraint ensures the FIFO processing of events within a topic queue. This means that on a $N$ processor machine with $N$ scheduling threads that are constantly running no topic can get a share greater than $1/N$.

Assume on an $N$ processor machine that the share allocated to topic $i$ is $0 \leq z_i < 1/N$ and that $s_i$ is the number of messages arriving at the topic queue per second. The question we wish to address is how many messages $r_i$ will actually be serviced for topic $i$. The number of events serviced for a topic can be viewed as containing two components: the rate guaranteed by the share and any spare capacity that a topic can take advantage of. We assume that the time taken to handle events in distinct queues is the same, adjusting the following to take this into account would be straightforward but would complicate the derivation. We denote the total maximum achievable rate by $R_{max}$ messages per second (msg/sec). $R_{max}$ is a function of system settings, but for the moment we consider it as a constant. In Section 4 we give heuristics for deriving unknown $R_{max}$ from known ones.

For example, suppose we have three topics $T_0$, $T_1$ and $T_2$ with shares $z_0 = 50\%$, $z_1 = 30\%$ and $z_2 = 20\%$, suppose further that we know that $R_{max} = 10,000$ msg/sec. What values are received for the three topics if $s_0=4,000$ $s_1=6,000$ and $s_2=5,000$ msg/sec? It is clear that $r_0$ is 4,000 msg/sec because topic $T_0$ is sending less than its share. That leaves 6,000 msg/sec to be shared amongst $T_1$ and $T_2$. Both $T_1$ and $T_2$ will get their guaranteed share (3,000 and 2,000 msg/sec respectively) plus some fraction of the spare capacity (1,000 msg/sec) that $T_0$ has reserved but is not using. The unreserved capacity is shared proportionally to the shares $z_1$ and $z_2$, so $T_1$ gets 60% and $T_2$ 40% meaning that $r_1=3,600$ and $r_2=2,400$ msg/sec.

We define $u_i$ to be the fraction of resources that a topic is attempting to consume and $x_i$ to be the ratio of $z_i$ to $u_i$.

$$u_i = \frac{s_i}{\sum s_j}, \quad x_i = \frac{z_i}{u_i}$$

An $x_i$ value of 1 means that a topic is attempting to use exactly what it has reserved. A value greater than 1 means that it has unused capacity and a value less than 1 means that it is trying to use more than its...
has reserved, i.e. it is trying to take advantage of any unused capacity.

For a given setting of $z_i$, $s_i$ and $R^{max}$ the expected receive rate $r_i$ is:

$$r_i = \text{Min}(s_i, z_i, \frac{R^{max} - \sum_{j \geq x_i} r_j}{1 - \sum_{j \geq x_i} z_j})$$  \hspace{1cm} (1)$$

This states that the share that a topic will get is its adjusted share of what is left over after topics with higher $x_i$ have been allocated. The amount of additional unused capacity available to a topic is that which is left over by topics which are less speculative than itself. This is equivalent to the Weighted Max-Min fairness allocation developed for the Available Bit Rate (ABR) service within ATM [6] where $u_i$ corresponds to the demand or rate and $z_i$ to the normalized weight.

The model described here does not take into account many practical issues that we know influence the performance of the scheduler. For example, the current implementation of the scheduler uses a lock-free concurrent priority queue to determine which topic should be scheduled next. The priority queue orders the topics by the amount of remaining credit that they have. For reasons of efficiency the credit is quantized into $M$ buckets such that all credit within a given range are given the same priority. In consequence, it may be that a given topic is not chosen to be scheduled although it should be according to the model because it shares the same priority level with topics with lower levels of credits. Topics within a given priority level are serviced in FIFO order so topics at the same priority level get an equal share while they remain at that level. A topic with less credit will be moved to a lower priority level quicker than one with more. The extent to which this effect reduces the fidelity of the model depends on the number of topics, the likelihood of their being in the queue at the same time and the size of the buckets. In Section 3 we evaluate the fidelity of this model.

3. Evaluation of the Scheduler Model Fidelity

The fidelity of the model to the real implementation of the scheduler is measured by comparing the throughput $o_t$ obtained on each of the $t$ topics with the corresponding throughput calculated by the model. We quantify the difference between the $t$-dimensional vector of rates $\vec{r} = (r_1, r_2, \ldots, r_t)$ predicted by the model and the vector of actual rates $\vec{o} = (o_1, o_2, \ldots, o_t)$ obtained in the experiment by calculating the euclidean distance between these two vectors $e = ||\vec{r} - \vec{o}||$.

The value $e$ by itself is not a good indicator of the performance of the model, because it depends on the magnitude of $\vec{r}$ and $\vec{o}$. To provide some meaningful reference, we compared this distance to the average distance between two randomly chosen points in the same vector space.

In a setup with two topics, and a total aggregate throughput $R^{max}$, clearly all possible configurations are on the line between $(0, R^{max})$ and $(R^{max}, 0)$ (with the two extreme settings being one topic getting nothing and the other all the throughput). Evaluating the average distance between each point on the line and a random point on that same line yields an average error of $e_{\text{rand}} = \sqrt{\frac{R^{max}}{2}}$. This is a lower bound for $e_{\text{rand}}$, because as we add dimensions to the problem the average error increases.

Thus our final measure of fidelity is defined as $e/e_{\text{rand}} \times 100\%$, with a value of 0 being a perfect match between the model and the implementation behavior, and 100 indicating that our model does not reflect at all what the scheduler is doing (i.e., our model is only as good as a random predictor).

In order to check how well the model described in Section 2.2 fits the actual behavior of the scheduler, we perform a number of experiments that cover a range of system configurations. The parameters we vary are:

- Total aggregate throughput $T$: The sum of the throughput being sent over all topics. This value is important with respect to $R^{max}$. If $T \ll R^{max}$ each topic should be getting exactly what it asks for, as there is enough capacity to satisfy all demand. When $T > R^{max}$ the scheduler assigns resources according to the shares of each topic. If $T \gg R^{max}$, collateral effects are expected to appear, as the system is highly saturated.

- Number of topics: In general, the greater the number of topics, the more unaccounted overhead there is in the scheduler. We test for 10, 20, 50 and 100 topics.

- Topic shares: The shares are randomly assigned to topics, having as only constrain that they must sum to 100% and that no topics gets more than 1/$N$ of the share where $N$ is the number of processors.

- Topic sending rates: Sending rates are randomly assigned to topics, constrained by the total aggregate throughput of the setting, but irrespective of
the share of that topic.

We produce 50 different randomly generated topic settings for each total throughput (30,000, 100,000 and 200,000 msg/sec) and number of topics (10, 20, 50 and 100), giving a total of 600 scenarios under test. Each experiment is performed as follows: a publisher application runs on one machine and a subscriber application runs on another, both using the same setting. Each machine is an IBM HS20 blade with 2 Intel(R) Xeon(TM) 3.20GHz 64-bit processors with Hyperthreading enabled and 2 Gbytes of RAM. The machines are connected directly using a Gigabit Ethernet switch. The message size is 128 bytes, so the network bandwidth is not an issue even at the highest sending rates. Two scheduling threads are run on the two core machine.

![CDF of the relative error for total aggregate throughput](image)

Figure 1. CDF of the relative error for total aggregate throughput of 100,000 msg/s depending on the number of topics, for 2 scheduler threads.

The results in Figure 1 show that the model reflects the behavior of the actual system with a very high degree of accuracy. As explained in Section 2.2, the larger the number of topics, the lower the fidelity. But even for 100 topics, in the worst case the prediction is 10 times better than the reference. For 10 topics, almost 90% of the settings’ behavior is described 1000 times more accurately by our model.

4. Estimating the Expected $R^{\text{max}}$ for a given Setting

4.1. General Approach

Section 3 showed that we can give a precise prediction of the likely achieved throughput of topics with arbitrary share and sending rates if we know the maximum achievable throughput $R^{\text{max}}$ at that setting, as described by Equation 1. However, $R^{\text{max}}$ itself varies with the system settings. For example, we would not accept the throughput achievable through a single topic queue to be the same as through a million topic queues. The $R^{\text{max}}$ value at a given setting is determined by the interaction between many parameters at many levels in the runtime environment, most of which are out of the control of the middleware. As such, it is almost impossible without calibration to say a priori the likely $R^{\text{max}}$ value at a given setting.

For example, for 10 topics with a total aggregate sending rate of 100,000 msg/sec, we observed an achieved throughput ranging from 45,000 to 90,000 msg/sec for arbitrary chosen settings. It is always possible to measure the $R^{\text{max}}$ for a given setting, but it is impractical to measure it for every possible combination as the number of possible distinct permutations of shares and sending rates rises factorially with the number of topics. Instead, we assume a calibration period in which a relatively small number of settings are tested to calculate their $R^{\text{max}}$. We then attempt to extrapolate from these known values of $R^{\text{max}}$ the value for some arbitrary setting and then use that estimated $R^{\text{max}}$ to predict the actual throughput of each topic in that setting. A given setting for a system with $t$ topics is completely defined by a vector of pairs: $\{ (z_0, u_0) \ldots (z_{t-1}, u_{t-1}) \}$, where $z_i$ is the fraction of the scheduler allocated to topic $i$ and $u_i$ is the fraction of the total number of events that are to be sent on this topic.

During the calibration period a certain number of such settings and their corresponding $R^{\text{max}}$ are obtained. We denote the $k^{\text{th}}$ setting as vector $V_k$. Hence the result of the calibration period is a set of pairs of the form $(V_k, R^{\text{max}}_k)$. We call this set $C$.

4.2. Closeness Hypothesis

Our intuition is that settings which are ‘similar’ are likely to have ‘similar’ $R^{\text{max}}$. We test this hypothesis by measuring the $R^{\text{max}}$ of every point in a two topic space in which the sending rate of each topic is varied from 1000 to 100,000 msg/sec by increments of 1000 and the share is varied from 1% to 100% by increments of 1%. This creates a space containing 5000 distinct combinations. We show in Figure 2(a) the effect on the measured $R^{\text{max}}$ as we move between adjacent settings. Figure 2(a) shows strong affinity between neighboring points but also that the relationship is not simple. Figure 2(b) shows in more detail the dependency of $R^{\text{max}}$ on the settings in a 2-topic space. Axis $x$ and $y$ show the throughput and share for the first topic (the second topic gets the remaining part of both settings). The vertical axis shows the achieved $R^{\text{max}}$. Note that when both topics produce similar throughput, $R^{\text{max}}$
equals to the total sending rate. This is also the case when the topic with higher throughput gets the higher share.

Our general approach to determine the $R_{\text{max}}^u$ of an unknown setting $V_u$ is to use the weighted average of the points in $C$, where the weights are a function of ’similarity’. This process has form, for some general weight function:

$$R_{\text{max}}^u = \sum_{(V_i, R_{\text{max}}^i) \in C} \text{weight}(V_i, V_u, C) R_{\text{max}}^i$$ (2)

We quantify the similarity between settings by measuring the distance between them. First we define the distance between the share/sending rate pairs $p = (z, u)$ that make up each setting as:

$$\text{pair\_dist}(p_i, p_j) = \sqrt{(p_i.z - p_j.z)^2 + (p_i.u - p_j.u)^2}$$ (3)

As vector settings may have distinct order (due to a different number of topics), we add sufficient $(z = 0, u = 0)$ pairs to the lesser one to ensure that the order is identical. We define the $k^{th}$ topic setting pair within a vector as $V(k)$.

We then define the distance between two vectors as the minimal value of the sum of $\text{pair\_dist}$ such that each pair $(z_i, u_i)$ in each vector is matched with exactly one in the other vector. Note that the first topic pair in a given setting may not be similar to the first topic pair in another setting, but may exactly match the, say, second topic pair in that setting. The distance $\text{dist}(V_i, V_j)$ is then the solution to the following optimization problem:

$$\text{dist}(V_i, V_j) = \text{Min}\left[ \sum_{0 \leq k, h \leq t-1} \text{pair\_dist}(V_i(k), V_j(h)) \right]$$ (4)

where each $k$ and $h$ may be used only once. Finding the minimal distance between two such settings is known as minimal perfect bipartite matching in graph theory and algorithms exist which are polynomial $O(n^3)$ with the number of vertices [7]. Figure 3 illustrates a bipartite graph with a minimal match. $X$ and $Y$ are two different settings, the vertices are the topic setting pairs and the edges are euclidean distances between pairs.

![Figure 3. Definition of Minimal Bipartite Matching](image)

### 4.3. Example Predictor

There is an arbitrary number of possible techniques by which the distances between the vectors in Equation 4 can be converted into weights to estimate $R_{\text{max}}^u$ with Equation 2. We motivate the general approach...
Improvement estimating $R_{\text{max}}$ (msg/sec)

**Figure 4.** $R_{\text{max}}$ estimator improvement over the average when all configurations are tested.

by defining one such method, $n\text{ClosestPredictor}$, in which we take the average of the $R_{\text{max}}$ values of some number $n$ of closest points. First we test this predictor using the small space described in Section 4.2. We estimate the $R_{\text{max}}$ for each setting using the $R_{\text{max}}$ obtained in the experiment for all other settings and compare the resulting prediction against simply using the arithmetic mean of all other settings $\text{meanPredictor}$. Figure 4 shows that this predictor does much better than simply taking the mean. Positive values indicate how much better a given predictor is than $\text{meanPredictor}$, negative values indicate that $\text{meanPredictor}$ is better. On average the $n\text{ClosestPredictor}$ outperforms the $\text{meanPredictor}$ by 5,000 msg/sec, which is 5% of the total attempted sending rate.

In order to test whether the prediction would still work for a larger topic space and a small number of points to calibrate with, we perform an experiment on a data set in which the sending rates and shares are arbitrarily chosen for 10 topics. The data set consist of 1000 different settings. The experimental setup is as described in Section 3. Figure 5 shows that the difference between the error estimating $R_{\text{max}}$ of $\text{meanPredictor}$ and the error of $n\text{ClosestPredictor}$. For the first few settings tested, the $n\text{ClosestPredictor}$ performs similarly to the $\text{meanPredictor}$. However, a few hundred settings provide enough information for the $n\text{ClosestPredictor}$ to clearly outperform the mean. Adding new settings improves the accuracy of the predictor, although the improvement slows down after a while. Figure 5 shows that even for a large setting space and a small number of calibration points the $n\text{ClosestPredictor}$ performs better than the mean. This is consistent with the close hypothesis described in Section 4.2.

**4.4. Discussion**

It is entirely possible than other predictors would give better results than that described in the previous section. Our current approach is to try a range of predictors over a data set, identify that which is giving the most accurate results and then use that as the predictor within the running system. Our intention is to refine the model adaptively as new information is gathered from the running system.

This type of ad-hoc model is analogous to those used in Time Series Analysis [8]. For example, in an Auto Regressive model (AR) in addition to assuming the preservation of the mean and variance we assume that there is some correlation between a value at time $Z_t$ in the sequence and those at time $Z_{t-k}$ and attempt to determine those correlations. Determining which model and which parameters to use for a given data set (model-identification) is not an automatic process. Some hints can be determined from examining the covariance matrix but for a general ARMA model many different parameters may fit the same data set equally well. As all of them cannot be correct (they will give different predictions) intuition is required to select amongst them.

We are currently exploring the use of techniques such as [9] to divide the calibrated points into distinct clusters, such that an unknown setting is deemed to belong to an identified cluster and that the $R_{\text{max}}$ is then interpolated from the values of that cluster. As [9] points out, which clustering techniques to use is also an art rather than a science.

**5. Related Work**

Most of the distributed real-time literature has focused on the problem of scheduling of tasks such that deadlines are met [10]. In the middleware literature QoS requirements have been mostly treated as
a discovery problem where the end-points with the appropriate features have to be put in contact [11], assuming enough resources to satisfy the needs, but without specifying the mechanisms to actually enforce the QoS requirements. Our approach is closest to the adaptive bandwidth reservation schedulers presented in [12]. Reservation based schedulers reserve a fraction of the total CPU to each task, such that the bandwidth given to a task is enough to fulfill its timing constraints. In [12] the authors propose to use adaptive reservation for the tasks, such that each allocated fraction of the CPU is determined by a closed control loop. The authors of [13] also present an on-line approach to adapt the QoS of distributed real-time systems, and use distances between states to define the optimal operational point. Control theory is used to bring the system to a desired point of operation, while maintaining some system performance level. In contrast to our approach, they assume a finite number of input states for the system. A supervisor can bring the system from any of those input states to a stable one, chosen as the closest state given some distance metric between the state vectors.

6. Conclusion

We have outlined an approach which allows a prediction to be made about the number of events per second that can be handled at an event queue using a scheduler that allocates shares to queues. We have compared the predictions made by this approach against the actual achieved throughput on a publish/subscribe system built using this scheduler. Our first results are encouraging, showing that we can model the behavior of the scheduler with great accuracy, and use that to estimate the system response for new settings based on the data obtained during a system calibration phase. This capability makes it possible to find a system configuration for a given desired behavior. In order for our approach to be useful the predictions have to be robust, i.e. not overly sensitive to the assumptions made during calibration, e.g. total sending rate used. Currently we are testing the robustness of the predictions. Future work will explore how the predicted throughput and delay at many distinct staged event queues, as in [2], can be combined in order to allow probabilistic guarantees to be given about the end-to-end latency.

References