Vrije Universiteit Amsterdam
Faculty of Sciences
Warsaw University
Faculty of Mathematics, Informatics and Mechanics

Urszula Herman-Iżycka
Student number: 1544071

Flash Crowd Prediction

Master's Thesis
in COMPUTER SCIENCE

Supervisors

Dr. Guillaume Pierre
Drs. Michał Szymaniak
Vrije Universiteit Amsterdam

Dr. Piotr Chrząstowski-Wachtel
Warsaw University

July 7, 2006

Abstract

Sudden surges of traffic, also known as flash crowds, present a significant problem to Web sites. Current systems deal with flash crowds by offloading a portion of the Web site load to a content delivery network. However, it is hard to determine in advance when the offloading should start. As a result, most systems react only after detecting a flash crowd during its initial phase. In this thesis, we implement a system capable of predicting rather than detecting a flash crowd. The prediction is based on linear regression analysis of the system load over time. We demonstrate that it is possible to configure our prediction system such that it achieves reasonable results on all the studied real-world traffic load traces.

Contents

Chapter 1

Introduction

The continuous growth in the number of Internet users often results in popular Web sites becoming overloaded. This might happen, for example, when a Web site is linked by some very popular news Web site such as Slashdot [12], or when its address is advertised to a wide public in the media [1, 2]. In both cases, the number of requests received by the Web site grows rapidly, causing the server's capacity to soon become exceeded. Overloaded Web sites service as many requests as possible (usually with very low performance), and simply drop the remaining ones. Such events are often referred to as Slashdot effects, hot spots, or flash crowds.

Flash crowds present a significant problem to Web site owners. In the case of commercial Web sites, a flash crowd can lead to severe financial losses, as clients often resign from purchasing the goods and search for another, more accessible Web site. However, non-commercial Web sites can also experience flash crowds. For example, an unfrequently visited government Web site might be saturated by client requests when it is used to announce important administrative regulations, as a great number of citizens will then attempt to retrieve the regulations content. In general, it is impossible to predict which sites shall be subject to flash crowds, and so each Web site should be ready to handle them.

The research community has investigated flash crowds for several years. A widely adopted solution consists of offloading a portion of the Web site load to some distributed infrastructure such as a content delivery network (CDN) [4]. In that case, the Web site replicates its content to a number of CDN servers, and starts redirecting the clients to these servers when the load becomes too high. This effectively increases the client-serving capacity of the Web site by that of the CDN, which enables the Web site to service all the clients with a good performance.

The problem with offloading is that the Web site has to decide when exactly the offloading should begin. If it starts too early, then the CDN capacity will be utilized unnecessarily, resulting in higher maintenance costs of the Web site. On the other hand, if the Web site begins offloading too late, then some of the clients shall still experience the reduced Web site performance during the initial flash crowd phase. Worse yet, replicating Web site content to a CDN during a flash crowd puts additional stress on the Web site, thus reducing its performance even further and causing the replication itself to last much longer compared to replication done in advance.

Ideally, a Web site would start offloading right before a flash crowd begins such that the CDN capacity is utilized optimally. To this end, the Web site would have to *predict* each flash crowd before it actually starts. However, current systems tend to merely *detect* flash crowds during their initial phases rather than predict them in advance. The classical flash crowd detection schemes continuously monitor the Web site load to detect that it has almost approached the Web site capacity. The inherent problem is that such a solution is very sensitive to the setting of the offloading threshold. If that threshold is set too high, then the Web site is informed about upcoming flash crowds at a very short

notice. In theory, the warning time could be longer if the threshold was set lower. This, however, would often result in numerous false alarms generated when the Web site load becomes high enough to exceed the threshold, but at the same time it is not high enough to be classified as a flash crowd.

A recent paper on flash crowd prediction suggests that Web sites can predict upcoming flash crowds by applying simple statistical techniques to the history of recently observed Web site load [6]. More specifically, a Web site can use linear regression to identify the ascending trend in its load, and verify whether the load anticipated in the near future might be classified as a flash crowd. The authors demonstrate that their solution offers an excellent balance between accuracy and simplicity, and that it succeeds at predicting a number of flash crowds. However, although the theoretical findings presented in the paper seem promising, one needs to address a number of additional problems before exploiting these findings in an actual flash crowd prediction system. For example, our experience with several real-world flash crowd traces indicates that the proposed flash crowd definition does not reflect the human intuition of what should be considered a flash crowd. Worse yet, it does not lead to correct Web site actions in terms of activating and de-activating the CDN infrastructure, causing the prediction algorithm to be of little practical use.

This thesis proposes a flash crowd prediction system built around an alternative flash crowd definition. Our definition aims at usability: an event should be characterized as a flash crowd if and only if the Web site needs to exploit the CDN servers in order to sustain it. Our prediction system combines that definition with the original prediction algorithm based on linear regression to decide when to activate and de-activate the CDN infrastructure.

The prediction system has the form of a pipeline, which takes the client request rate as its input and outputs decisions to activate or de-activate the CDN infrastructure. The pipeline consists of four components. First, the original request rates are aggregated by a pre-processing component. Next, they are passed to two other components, responsible for prediction and detection of flash crowds, respectively. The output of these two components is combined by a post-processing component, which makes the final (de)activation decision. Our experience indicates that combining prediction and detection is necessary to achieve good performance, as each of them recognizes different flash crowd phases.

We test our prediction system based on a number of real-world flash crowd traces of different types. As one could expect, the system performance depends greatly on both the trace characteristics and the system configuration. However, we demonstrate that there exists a set of configuration parameters that allows for achieving reasonably good performance on all the traces. We also show that adjusting the system configuration to individual trace characteristics allows for improving the performance of our prediction system even further.

The remainder of this thesis is structured as follows. Chapter 2 presents general flash crowd characteristics along with a description on how current systems deal with flash crowds. Chapter 3 discusses the formal flash crowd definition. Chapter 4 treats of the prediction system architecture. Chapter 5 describes the evaluation criteria followed by the performance results. Finally, Chapter 6 concludes.

Chapter 2

Related Work

A flash crowd is a sudden growth of request rate experienced by a Web site. Since Web sites are seldom prepared to service requests at high rates, the site performance during a flash crowd is typically degraded. This is undesirable, as low performance of a Web site discourages clients from using it, which usually translates into bad site reputation and financial losses. To prevent such situations from happening, a Web site needs to implement some techniques for flash crowd handling, such that it can offer good performance to its clients even when receiving requests at very high rates.

This section gives the foundation to the entire thesis by summarizing a number of research efforts dealing with the issue of flash crowds. In principle, these efforts fall into three categories. The first category defines flash crowd properties, which we analyze to better understand the phenomenon of a flash crowd. The second category builds on the first one by proposing concrete systems for flash crowd handling. These systems demonstrate how a Web site can adjust its operation and continue to efficiently service its clients under severe conditions. The third category shows that flash crowds can be predicted, which potentially enables the site to adjust its operation in advance, leading to better site performance. We build on the research presented in the three above categories and propose how to predict flash crowds more effectively than using earlier techniques.

As we present the research efforts, we use a unified terminology to avoid confusion. We refer to the server affected by a flash crowd as *origin server*. The remaining terms are introduced later as we need them.

2.1. Flash Crowd Characterization

Flash crowds might come to existence for various reasons. For example, some flash crowds, called Slashdot effects, occur when an origin server is linked by Slashdot [12] or any other popular news Web site. Typically, such linking results in a large number of Web clients accessing the origin server, which leads to a flash crowd lasting several hours [3]. The actual flash crowd duration is often correlated with the time for which the link to the origin server is visible on the front page of the news Web site.

The number of clients visiting the origin server during a flash crowd changes over time. Figure 2.1 depicts how the number of requests to some origin server evolves when that Web site is linked by two news Web sites. The origin server hosted the “Preserving the Information Ecosystem” publication, which attracted significant attention of the Internet users. The first vertical line denotes the moment when the publication was commented on in the Linux Today. The link to the origin server remained visible on the Linux Today Web page for about 12 hours. The second vertical line indicates the time when an editorial about the publication appeared on Freshmeat, which kept the editorial on its front page for several days.

As can be observed, the Linux Today peak is lower than that for Freshmeat. This is most likely

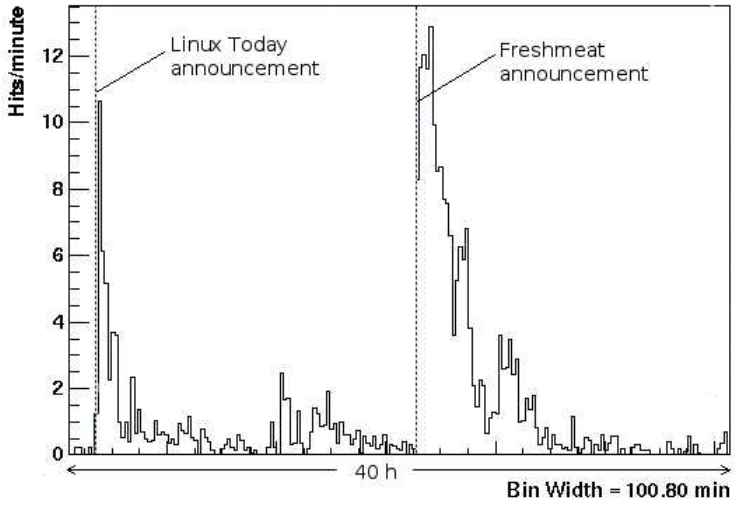


Figure 2.1: Server hit statistics for the Adler publication (adapted from [3]).

because Freshmeat has more readers than Linux Today. Also, in both cases, the number of requests gradually decreases after the initial peak, and reaches its usual level shortly after the link to the origin server is removed from the front page of a respective news Web site.

Another publication, “An Ode to Richard Stallman”, also attracted the interest of Internet users. The hit rate evolution for this article is depicted in Figure 2.2. The publication was initially read only by the subscribers of the Redhat and LNXY mailing lists. Later, Linux Today and Slashdot announced the publication, which is marked by the two vertical lines. The first line corresponds to the Linux Today publication and the second one to the time when the article was announced on Slashdot.

As can be observed, announcing the publication on Redhat and LNXY caused only a slight increase in the request rate, and the actual flash crowd started when the article was linked by Linux Today and Slashdot. Interestingly, the form of the announcement influenced its resulting traffic surge. Linux Today published a complete article text, whose copy was hosted on the Linux Today server. Slashdot, in turn, published only a hyper link to the original article hosted on the origin server. As a consequence, announcing the article on Slashdot caused a distinctly larger traffic surge compared to publishing the article text on Linux Today. Another important observation is that the traffic surge was sudden, but not instantaneous. The request rate increased from about 30 requests per minute up to over 250 requests per minute within 15 minutes. This observation underlies the assumption that one can predict flash crowds by analysing the trends in request rates.

Flash crowds are somewhat similar to denial of service (DoS) attacks, as both of them cause a sudden growth in request rates. However, while requests arriving during a flash crowd should be processed, those received during a malicious attack should be rejected. To distinguish between flash crowds and DoS attacks, Jung *et al.* characterizes both of them based on traffic patterns, topological client distribution, and file reference characteristics [9]. The authors identify three major differences between flash crowds and DoS attacks. First, flash crowds cause *per-client* request rates to decrease, as the origin server can handle fewer requests per client, which causes each client to reduce its request rate. Second, the topological distribution of clients during a flash crowd is preserved in the sense that clients typically originate from the same networks as before the flash crowd. Third, the popularity of files requested by the clients follows the Zipf-like distribution [16].

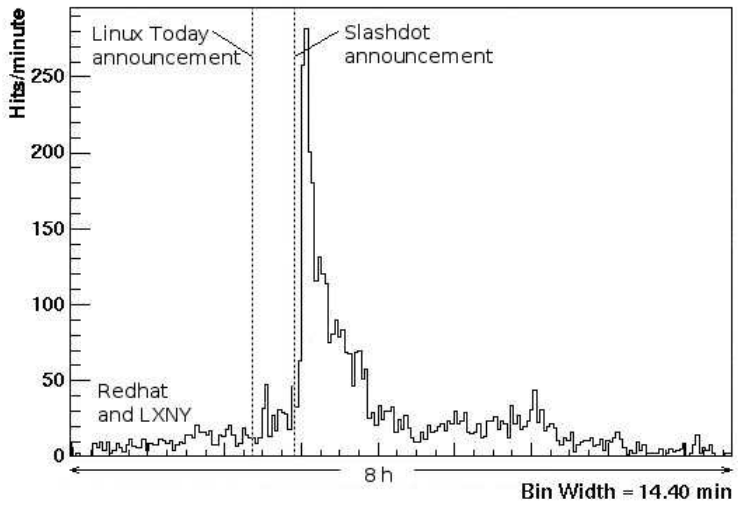


Figure 2.2: Early phase of the flash crowd.

On the other hand, DoS attacks exhibit different properties. First, they cause per-client request rates to remain stable or even increase, as clients do not care about server responses and therefore do not adjust their request rates accordingly. Second, a large number of requests originates from randomly dispersed clients that have never been seen before. Third, the popularity of requested files does not follow Zipf-like distribution. Rather, only a certain small set of files is requested, many of which may even not exist on the server.

The above differences between flash crowds and DoS attacks can be used to devise strategies for detecting and ignoring malicious requests. However, they also contribute to our better understanding of what flash crowds are and how they come to existence.

2.2. Flash Crowd Detection and Handling

Detecting a flash crowd forces the origin server to adjust its operation such that it can continue servicing clients. This typically requires using some distributed infrastructure of *mirror servers*, which provide the origin server with additional hosting capacity. Mirror servers can originate either from a CDN [9] or from a community of contributed servers [8, 15]. The origin server typically exploits the capacity of mirror servers by using them to service some fraction of client requests. This section surveys various techniques for flash crowd detection and handling.

Before starting to adapt its operation, the origin server needs to actually detect a flash crowd. The simplest technique is to monitor the request rate, and initiate adaptation once that rate exceeds a certain threshold. The intuition behind this technique is that the high request rate can be treated as the indication of an upcoming flash crowd. In practice, however, using a single adaptation threshold is not enough, as request rates might oscillate around it, thereby causing frequent and unnecessary system adaptations.

To address the oscillation problem, several algorithms use a simple watermarking technique. In that case, the origin server defines two watermarks, high and low, and determines its state as follows:

Definition 2.1 *We say that a server enters the overloaded state, when the request rate reaches the high watermark. The server remains in overloaded state until the request rate drops below the low*

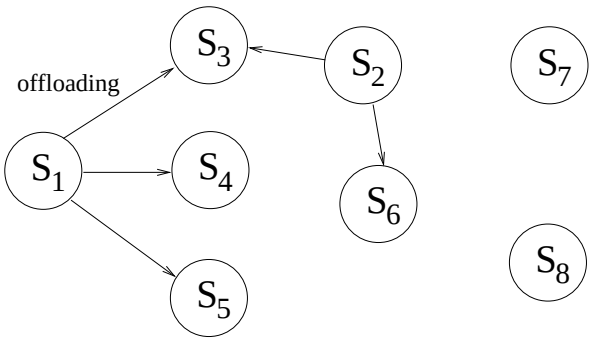


Figure 2.3: DotSlash architecture. S_1 and S_2 are origin servers. S_3,S_4,S_5 and S_6 are mirror servers (adapted from [15]).

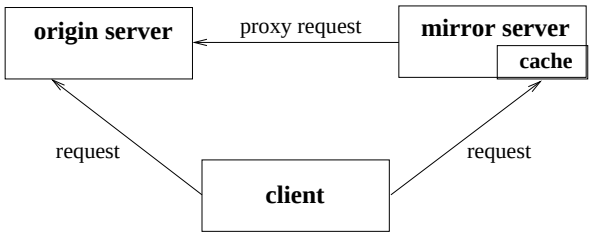


Figure 2.4: DotSlash HTTP redirection.

watermark. When the request rate reaches the low watermark, the server returns to its normal operation mode.

This watermarking technique is exploited in the DotSlash system to decide when to start delegating client requests to mirror servers [15]. When the origin server enters the overloaded state, DotSlash seeks underloaded mirror servers and redirects some of its requests to them (see Figure 2.3). The experimental results based on synthetic data indicate that DotSlash enables the origin server to service at least an order of magnitude more clients.

However, the problem with DotSlash is that it does not replicate its content to mirror servers in advance. Instead, the mirror servers act as caches, which fetch content from the origin server upon each cache miss (see Figure 2.4). Given that mirror servers are used to serve content only when the origin server is overloaded, they intuitively should not contact the origin server at all to avoid increasing the origin server load any further.

The limitation of DotSlash is addressed by Felber *et al.*, who propose a method to detect and avoid flash crowds proactively [8]. The main idea of the algorithm is to detect flash crowds gradually, and to adapt the server operation according to the current flash crowd phase. The adaptation strategies include redirection and replication, and the algorithm applies them based on the watermarking technique. As opposite to DotSlash, however, it exploits an additional "middle" watermark between the high and low watermarks, which is treated as a warning about a potentially upcoming flash crowd.

The three watermarks are used as illustrated in Figure 2.5. More precisely, exceeding the middle watermark is an indication of a possibly upcoming flash crowd. At that time, the contents of the origin server are replicated to mirror servers in anticipation of a flash crowd (time t_1). The content is being replicated as long as the request rate is between the middle and high watermark. However, the origin server does not redirect clients to the mirror servers yet. It starts doing it once the high watermark has been reached (time t_2). Analogous to what happens in DotSlash, repetitive changes in adaptation

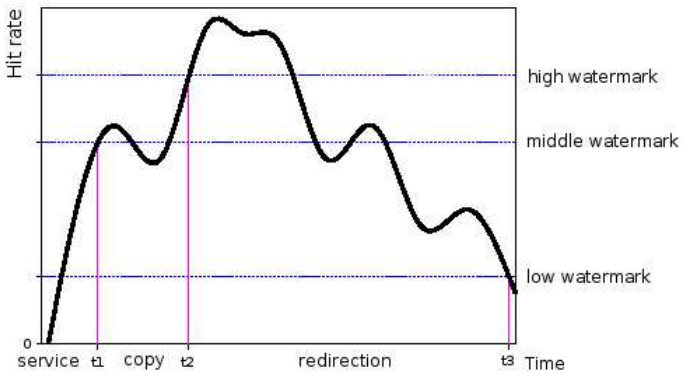


Figure 2.5: Felber algorithm specifics (adapted from [8]).

strategies are avoided by returning to the normal state only when the *low* watermark is reached again. This means that clients are redirected until time t_3. Interestingly, content replication can be recursive, with mirror servers replicating their content and redirect clients to other mirror servers. Still, the overall system capacity remains constrained by the initial redirection performed at the origin server.

A somewhat different adaptation scheme is exploited in RaDaR (Replicator and Distributor and Redirector), which has a broader goal than only providing rescue to the origin server in times of a flash crowd [10]. RaDaR is an architecture for a large pool of globally-distributed servers that continuously migrate and replicate content among themselves to optimize per-server request rates and per-client access latencies. The proposed hierarchical model resembles the hierarchy of the Internet, and implements a special distributed service responsible for content migration and replication (see Figure 2.6).

RaDaR is not specifically designed to handle flash crowds. Rather, its main functionality is to ensure efficient resource usage. However, the system is scalable and can redirect clients to their closest servers while considering the load of each server at the same time. Extending RaDaR with a module for predicting flash crowds could be a natural next step, as RaDaR already implements all the necessary adaptation techniques.

The continuation of RaDaR is the Application CDN [11]. ACDN extends RaDaR by the ability to deliver *dynamic* content. Also, it implements a better request distribution algorithm, which spreads the load more evenly across the servers and offers a better average access latency.

Similar to other systems, ACDN also defines two watermarks for request rates, but uses them in a different manner. In essence, any server with load above its high watermark receives no requests at all. If the load is below low watermark, in turn, the server gets all the traffic. Otherwise, servers are assigned weights according to their respective load. Also, when the average server load increases, access latencies become less important than load balancing.

The experimental results show that ACDN is able to efficiently distribute requests among its servers, especially because it focuses on load balancing instead of latency optimization once the average server load increases. However, gradual adaptation to changes in request rates might still turn out to be too slow to effectively deal with a flash crowd.

2.3. Flash Crowd Prediction

A recent study shows that flash crowds might not only be detected based on the observed workload, but that they can actually be predicted shortly before they start [6]. The authors evaluate different

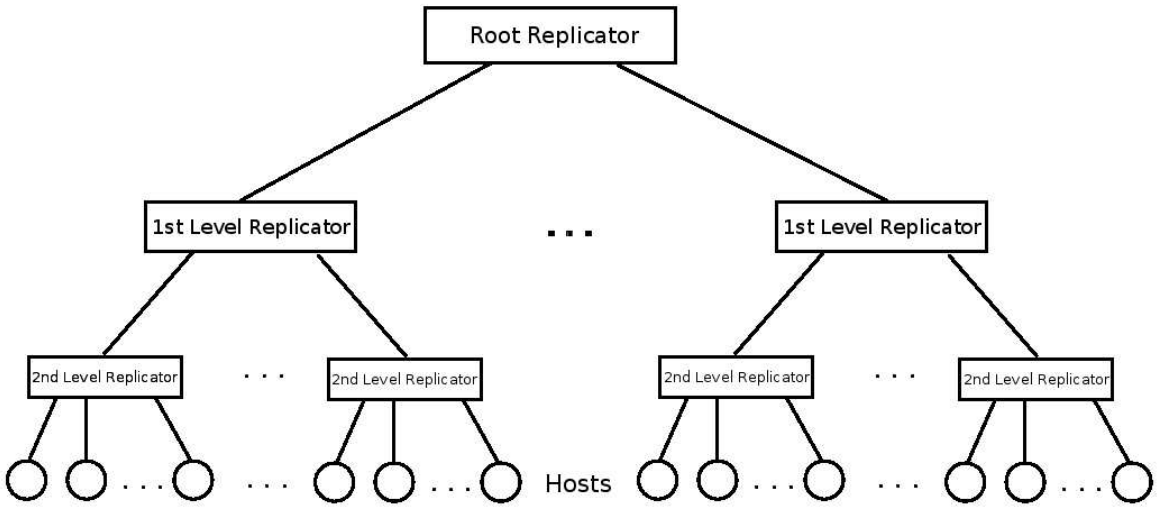


Figure 2.6: RaDaR replicator hierarchy. Tree leaves are servers, that provide content to the clients (adapted from [10]).

prediction methods based on several flash crowd traces, and demonstrate that linear regresion offers excellent balance between accuracy and simplicity. The results appear to be very promising and offer broad field for exploitation. We present these results in detail as they are the starting point for this thesis.

2.3.1. Definitions

The descriptive definition of a flash crowd is insufficient in order to conduct precise mathematical studies. Therefore, the authors mathematically define a flash crowd as follows. First, they define r_t as the number of page requests in a time slot, which is defined as the time unit. The slot size depends on the data structure or the desired reaction time. Second, they define H as the maximum capacity of the server. Finally, they define a flash crowd at time t as the event during which $r_t > H$.

However, large variations between samples make this definition impractical. Samples oscillate around H and the server would constantly change its state, making the whole system unstable. The authors therefore propose to aggregate the samples into windows of size W_d, such that flash crowds can be reliably detected. The resulting flash crowd definition is therefore:

Definition 2.2 *We say that traffic is experiencing a flash crowd at time t if the average request rate over* $[t - W_d, t]$ *is at least* H.

Whereas the above definition allows for a detailed flash crowd analysis to be conveyed, our experience indicates that it can be improved even further. In the next chapter, we discuss the implications of such definition for the detection algorithm in greater detail, and propose how to improve it.

2.3.2. Linear Regression

The prediction algorithm in [6] is based on the linear regression method. It allows one to predict the value of an independent variable at some future time t given a number of variable samples measured

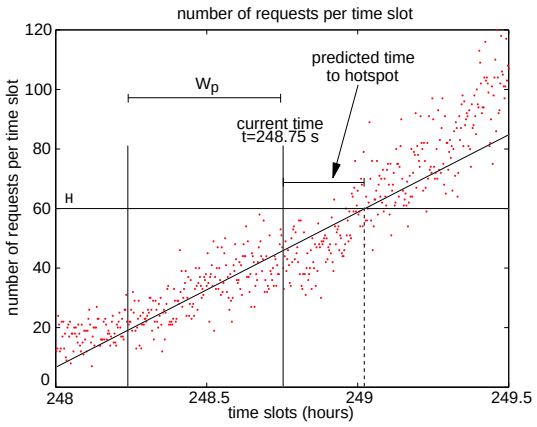


Figure 2.7: Linear regression and advance notice (adapted from [6]).

in the past.

The authors apply linear regression at any time t over a set of samples collected between time $t - W_p$ and t, where $W_p \geq 0$ is a *prediction window* parameter. The regression allows to predict the variable value at time $t + \tau$, where $\tau \geq 0$ denotes the *advance notice* parameter. Thus, the authors define a mapping from observations in the prediction window $[t - W_p, t]$ to a number $p_t \equiv p_t(\tau) \geq 0$ of predicted load value in the interval $[t + \tau, t + \tau + 1]$, starting τ time units in the future.

Linear regression uses the extrapolation of a least-squares linear fit to define p_t, i.e. $p_t = f_t(t+\tau)$, where the coefficients of $f_t(s) = a_t s + b_t$ are chosen to minimize the mean quadratic deviation over the window $[t - W_p, t]$, that is $\sum_{i=t-W_p}^{t} [f_t(i) - r_i]^2$.

2.3.3. Prediction Algorithm

The prediction algorithm works in two steps. First, it exploits linear regression to produce a sequence of advance notices for the request rate. Second, it verifies whether and when the request rate is expected to exceed H. The predictions are valid only when the results p_t are positive and finite. This means that the linear regression of request rates from the window $[t - W_p, t]$ (t representing the current time) should have a positive slope and intersect the H level at some time in the future (see Figure 2.7).

Reliable flash crowd detection requires analyzing general *trends* observed for groups of samples rather than making predictions in a few individual points. This means in particular that the size of the linear regression window W_p should be sufficiently large, as only then is it possible to isolate actual trends in request rates. Also, the window size determines the upper bound on the advance notice τ_{max}, beyond which predictions are too unreliable to be made.

The ultimate prediction algorithm can be defined according to when it raises flash crowd alarms. First, an alarm is set if the origin server is currently in a flash crowd. Second, an alarm is also set when the trend of advance notices isolated using linear regression over the $[t - W_p, t]$ window indicates that a flash crowd will occur at time $t_a \in [t, t + \tau_{max}]$. If neither a flash crowd is in progress nor is it predicted, then the alarm is not set.

2.3.4. Prediction Performance

The authors evaluate their predictor based on a real flash crowd trace collected for the server hosting the Winter Olympic Games Web site depicted in Figure 2.8. Predicting flash crowds in that trace is

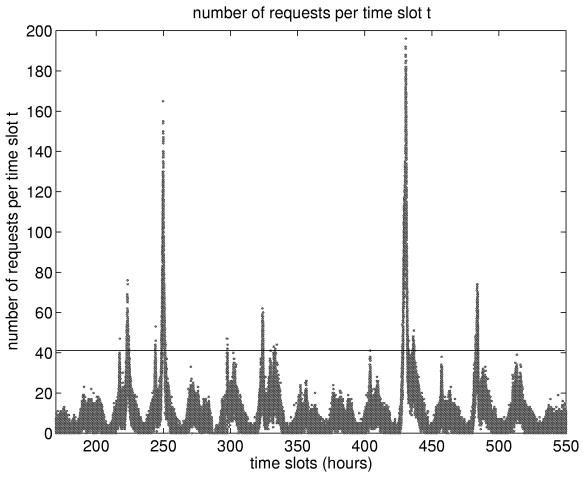
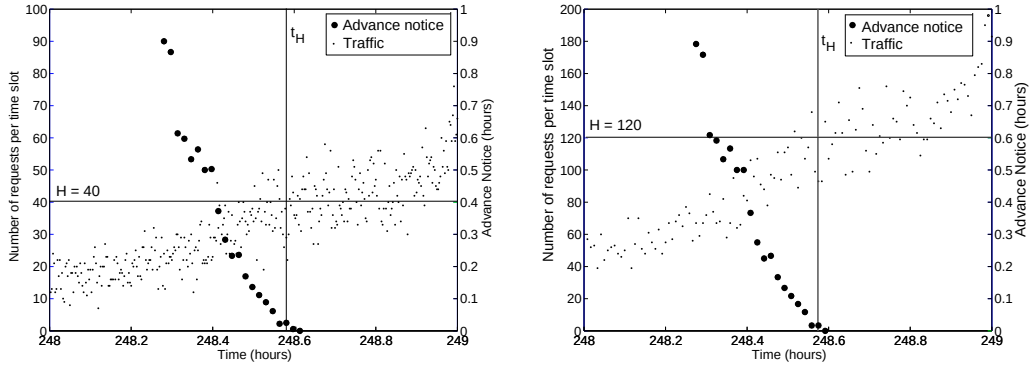


Figure 2.8: Winter Olympics trace with $H = 40$.



(a) 10 seconds per time slot

(b) 30 seconds per time slot

Figure 2.9: Sequence of advance notices with different time slot granularities.

challenging, as several peaks almost reach the H threshold, potentially leading to many false alarms.

Figure 2.9 illustrates how advance notice sequences can visualize the arrival of a flash crowd. The X axis denotes time, whereas the left hand Y axis specifies the number of requests per time slot. The right hand Y axis, in turn, shows advance notices at different moments in time. A well-predicted upcoming flash crowd is represented by a linearly decreasing sequence of advance notices. This means that they should be close to 0 at about the flash crowd arrival time (denoted as t_H in Figure 2.9), i.e. when the request rate averaged over the last W_d time units increases to H.

The effectiveness of prediction depends on the choice of parameters. One of such parameters is the maximum advance notice τ_{max}. Too small values of τ_{max} can cause the algorithm to return many false alarms, i.e. situations in which the flash crowd is predicted, but the load will eventually not exceed the H threshold. On the other hand, τ_{max} should not be too large, as it will then constraint the algorithm reactiveness (see figure 2.10). In the case depicted in Figure 2.9, the combination of τ_{max} of 5 minutes and the W_p of 15 minutes seems to be a good choice.

Another parameter is the granularity of samples. The authors demonstrate that it may greatly influence the prediction quality, and show that increasing the granularity accelerates the prediction algorithm without significant degradation of the prediction quality (see Figure 2.9).

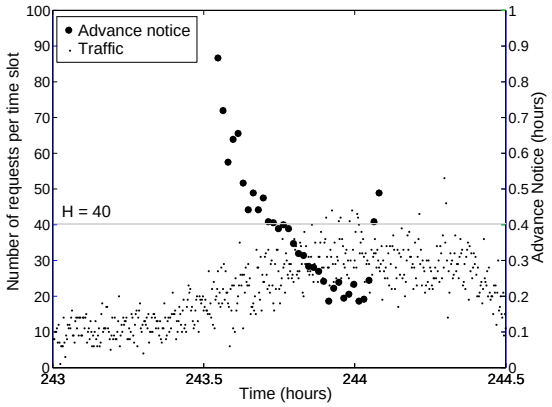
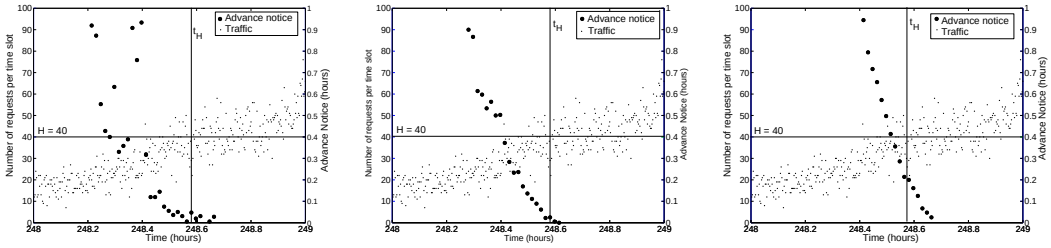


Figure 2.10: An example of a too large τ_{max}.



(a) $W_p = 10$ minutes (b) $W_p = 20$ minutes (c) $W_p = 45$ minutes

Figure 2.11: Prediction quality for different W_p.

A more sensitive parameter than data granularity is the prediction window W_p. The size of the window may vary, but only to a certain extent. A too small W_p can lead to erratic advance notice sequences, as the prediction algorithm is sensitive to local variations. On the other hand, prediction algorithm with a too large W_p uses unfresh data for the prediction. Figure 2.11 depicts how prediction performance depends on the window size.

2.4. Discussion

The discussed research efforts identify a number of properties exhibited by typical flash crowds. Some of them, such as gradual increase in the request rate, can be exploited to predict flash crowds shortly before they happen. Predicting a flash crowd could enable the origin server to adapt in advance, and not only when the flash crowd begins. This, in turn, is likely to result in a better server performance during the flash crowd.

The proposed prediction algorithm is based solely on linear regression. However, although it offers promising results, we are convinced that it can be improved further. The following chapters demonstrate how this can be achieved.

Chapter 3

Definitions

The overview of flash crowd characteristics helps to develop a good intuition about what flash crowds are and how they come to existence. However, deciding whether a given event is a flash crowd is still quite subjective. Therefore, to allow for systematic studies of flash crowds, a more formal definition is necessary. In particular, such a formal definition is essential when comparing the performance of different prediction strategies.

We propose a formal definition that aims at usability: an event should be characterized as a flash crowd if and only if the Web system needs to adjust its operation in order to sustain it. Note that this meta-definition rules out short-lived traffic overloads. We believe that the right reaction to these relatively frequent events is simply to serve the requests without adapting, as the cost of performing an adaptation would most likely exceed the benefits that adaptation would provide.

Our definition is a result of an evolutionary process. We start with the definition proposed in the original paper on flash crowd prediction described in Section 2.3. Then, we gradually improve the definition by eliminating its subsequent shortcomings.

Each definition improvement is illustrated based on a hypothetical flash crowd trace depicted in Figure 3.1 (graph presents the average request rate over $[t - W_d, t]$ for each t). As can be seen, there are six peaks of different kinds. Our aim is to present a definition that identifies two flash crowds and one short-lived overload, which we call a burst.

More specifically, peak 1 depicts a rather small and short-lived traffic growth. However, it is close to peak 2. Since peak 2 is a result of a long-lasting growth in the request rate, the system should adapt to handle it properly. Given that peak 1 is in close vicinity of peak 2, the adaptation should already start at the beginning of peak 1, and peaks 1 and 2 should be recognized together as a single flash crowd.

Peak 3 is also a short-lived increase in the request rate close to a large peak 4. Similar as in the case of peaks 1 and 2, peaks 3 and 4 should be handled together. However, these two peaks are followed by another long-lasting peak (5), which happens after a sudden decrease in the request rate (6). There are many reasons for such a decrease, such as a transient system failure caused by the flash crowd itself. We believe that such transient decreases should not affect flash crowd continuity. In particular, they should not cause the system to become unadapted again. Our definition should therefore consider peaks 3, 4, and 5 as a single long-lasting flash crowd.

Peak 7, in turn, is an isolated short-lived increase in the request rate. Although such a short-lived increase might degrade the system performance, it affects only a small number of clients for a short time. Hence, adapting to such short-lived increases would most likely result in wasting the system resources. Our definition should therefore identify peak 7 as a burst, and not as a flash crowd.

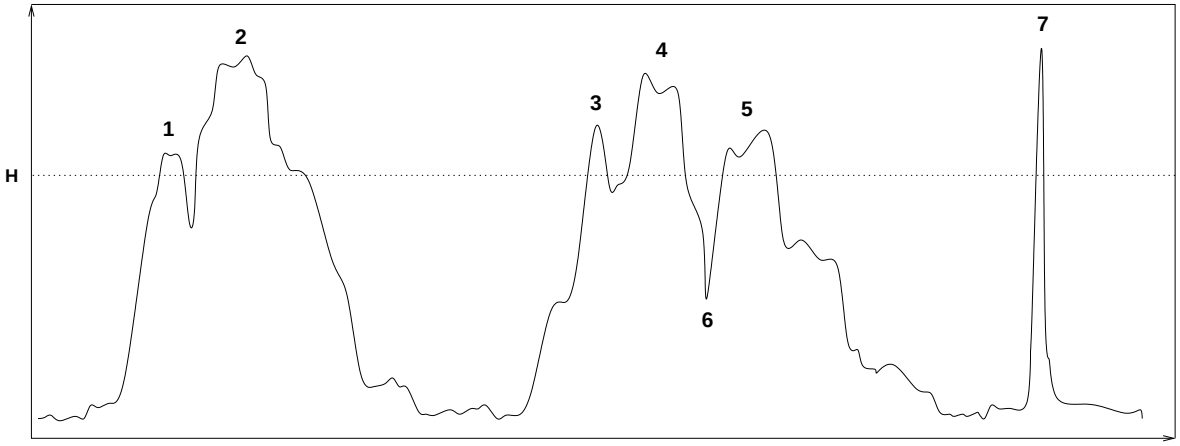


Figure 3.1: A hypothetical flash crowd trace.

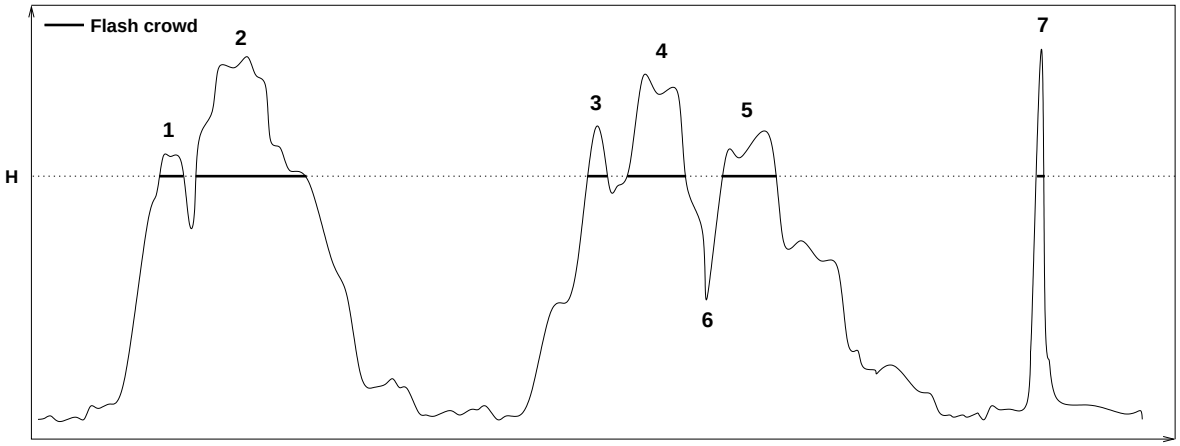


Figure 3.2: Flash crowds identified according to Definition 3.1.

3.1. Definition Improvements

The starting point of the definition evolution is the definition presented in Section 2.3:

Definition 3.1 *We say that traffic is experiencing a flash crowd at time t if the average request rate over* $[t - W_d, t]$ *is at least* H.

According to the original definition, our hypothetical trace contains six flash crowds (see Figure 3.2). In particular, it considers different phases of the two long-lasting flash crowds as independent events. This is because the request rate sometimes decreases below H for a couple of minutes only to increase above H afterwards, which might be caused by, for example, a transient system failure. However, a system relying on the current flash crowd definition would react to such oscillations by adapting continuously, which is likely to be very expensive.

We solve this problem by means of the classical watermarking technique, which is commonly used by other systems to handle oscillations in the request rate. We define a flash crowd level H_d (low watermark) at a certain fraction of the H threshold (high watermark). Also, we define three load zones:

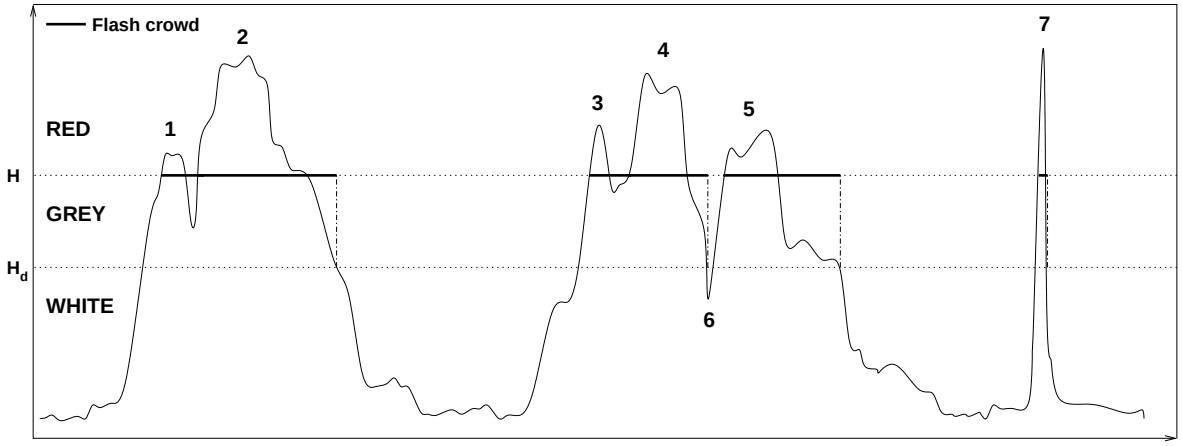


Figure 3.3: Flash crowds identified according to Definition 3.2 (zones).

- red: $r_t \geq H$,
- grey: $H_d \leq r_t < H$,
- white: $r_t < H_d$.

These definitions allow for refining the flash crowd definition as follows:

Definition 3.2 *The flash crowd is said to start at time t_{start} when the traffic enters the red zone, i.e. the average request rate over $[t_{start} - W_d, t_{start}]$ is at least H. It is said to stop at time t_{end} when it enters the white zone, i.e. the average request rate over $[t_{end} - W_d, t_{end}]$ is less than H_d.*

Introducing the watermarks results in aggregating closely situated peaks. However, as can be observed in Figure 3.3, the new definition identifies four flash crowds, still considering peak 5 to be a separate flash crowd. This is because the request rate decreases so much between peaks 4 and 5 that it crosses the low watermark. We intuitively call such short-lived decreases in the request rate as "negative bursts". A system relying on the current flash crowd definition would react to each such negative burst by de-adapting itself, believing that the flash crowd is over. On the other hand, it makes more sense to keep the system adapted during a negative burst to save on the cost of re-adaptation.

We refine our flash crowd definition to interpret negative bursts properly by introducing another parameter, W_e, which denotes the longest time for which the request rate can remain in the white zone during a flash crowd. In other words, a flash crowd is considered to be over only after the request rate has stayed within the white zone for W_e seconds, which leads to the following definition:

Definition 3.3 *The flash crowd is said to start at time t_{start} when the traffic enters the red zone, i.e. the average request rate over $[t_{start} - W_d, t_{start}]$ is at least H. It is said to stop at time t_{end} when it enters and remains in the white zone for more than W_e time units, i.e. $\forall_{s \in [t_{end} - W_e, t_{end}]}$ the average request rate over $[s - W_d, s]$ is less than H_d.*

As depicted in Figure 3.4, the last problem that needs to be solved is that the burst represented by peak 7 is still considered to be a flash crowd. The reason here is that our zone definition does not distinguish between flash crowds and bursts, as it does not consider the duration of an overload. We solve this problem by introducing a new parameter, τ_{fc}, which denotes the minimal time for which a flash crowd should last. The resulting final definition of a flash crowd states therefore as follows:

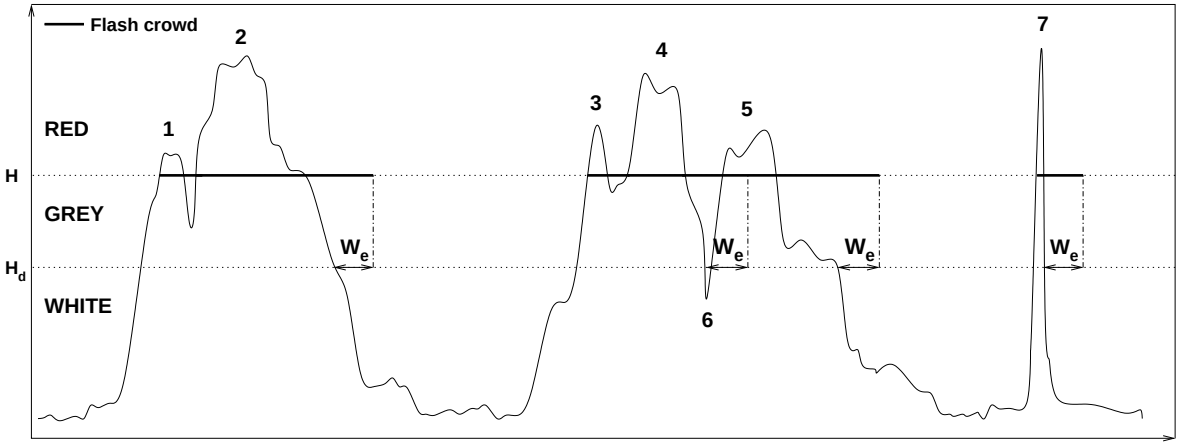


Figure 3.4: Flash crowds identified according to Definition 3.3 (zones and W_e).

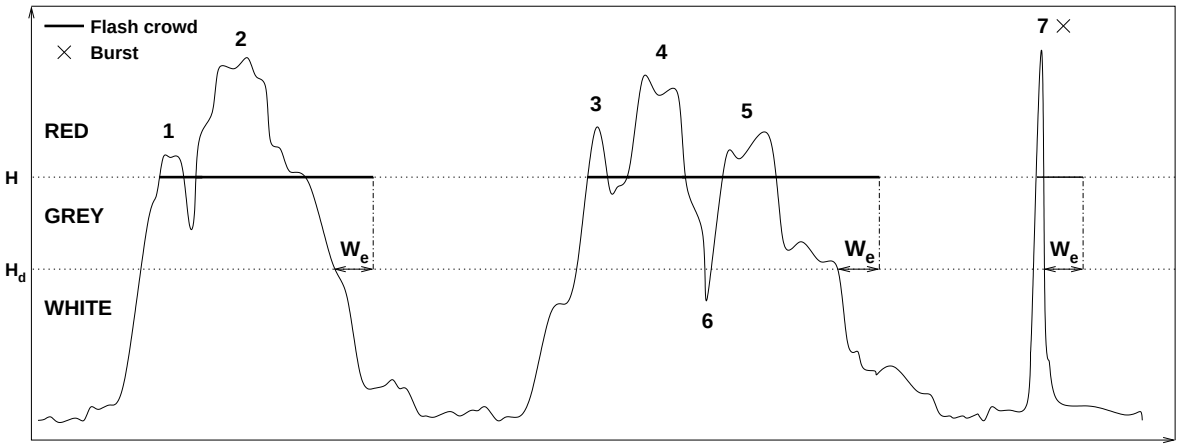


Figure 3.5: Flash crowds identified according to Definition 3.4 (zones, W_e and τ_{fc}).

Definition 3.4 *The flash crowd is said to start at time t_{start} when the traffic enters the red zone, i.e. the average request rate over $[t_{start} - W_d, t_{start}]$ is at least H. It is said to stop at time t_{end} when it enters and remains in the white zone for more than W_e time units, i.e. $\forall_{s \in [t_{end} - W_e, t_{end}]}$ the average request rate over $[s - W_d, s]$ is less than H_d. The difference $t_{end} - t_{start} - W_e$ should be greater than τ_{fc}. All other cases, where for time t the average request rate over $[t - W_d, t]$ is at least H are defined as bursts.*

3.2. Summary

The notion of flash crowd is by nature imprecise and so there is no ideal definition. However, we propose a formal definition that is driven by expected system reactions, and therefore approximates human intuition.

Our definition relies on a number of parameters. First, it properly interprets oscillations around the capacity threshold by means of watermarking. Second, it eliminates the problem with negative bursts by introducing the W_e interval during which the request rate should remain in the white zone

before the flash crowd is considered to be over. Finally, it distinguishes between bursts and actual flash crowds by means of τ_{fc}, which denotes the minimum duration of a flash crowd.

Introducing the formal definition enables us to evaluate the performance of our flash crowd prediction system. We discuss this system in detail in the next chapter.

Chapter 4

Prediction System

The goal of a good prediction system is to output accurate warnings of upcoming flash crowds. For a given time period, the prediction system should therefore detect the same flash crowds as those identified based on the formal flash crowd definition. The vital property of a prediction system, however, is that it issues warnings before flash crowds actually begin, thus allowing some preventive actions to be taken by the origin server to preserve high client-handing performance.

This section describes the architecture of our prediction system. We first discuss the system interface, and then explain its actual architecture in detail.

4.1. Functional Definition

As illustrated in Figure 4.1, our prediction system can in essence be seen as a black box which reads the request rates observed in subsequent time units, analyses them, and decides whether to activate or de-activate the infrastructure of mirror servers. The black-box approach hides all the details of prediction from the user and makes our system highly usable, as no further interpretation of its decisions is necessary.

More precisely, the input to the prediction system is a stream of (time, request rate) pairs. Each such pair denotes the request rate observed for a given moment, and all the pairs are sorted ascendingly according to the observation time. For each pair it reads, the prediction system returns its decisions in the form of (time, alarm) pairs, in which the "time" field corresponds to the time of the last observation result read from the input stream. The "alarm" field, in turn, can have either one of two possible values: "set", meaning that the origin server should switch to, or remain in the adapted mode, because of upcoming/lasting flash crowd; or "unset", which means that the origin server can switch to, or remain in the normal operation mode, because there are no upcoming/lasting flash crowds any more.

Deciding whether to activate or de-activate the server infrastructure is hard for two reasons. First, recall that our flash crowd definition distinguishes between two types of overloads: long-lasting, called flash crowds, and short-lasting, called bursts. The problem is that bursts are practically indistinguishable from flash crowds in their early phase. This means that the prediction system cannot differentiate between bursts and flash crowds when analyzing the request rates observed for subse-

Figure 4.1: Prediction system as a black box.

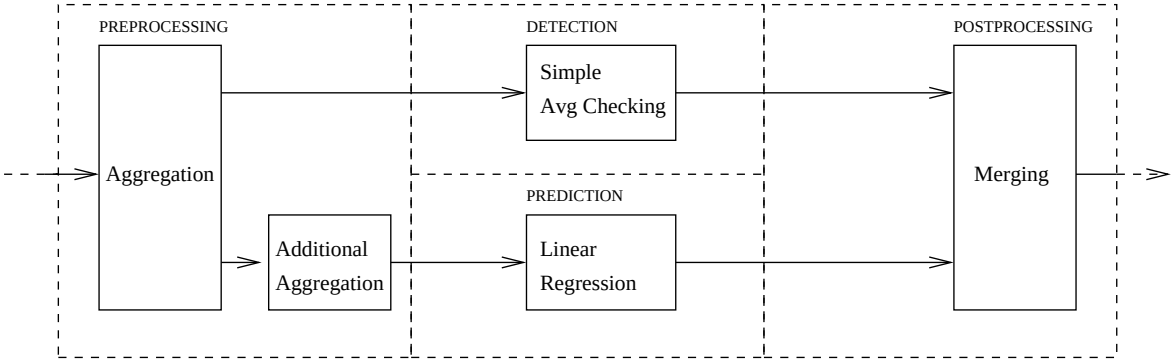


Figure 4.2: Detailed architecture of the prediction system.

quent time units. As a result, our prediction system might incorrectly decide to adapt the server infrastructure to handle bursts.

Another reason why making right activation decisions is hard is that request rates sometimes oscillate around the capacity threshold H. In particular, the request rates may remain just below that threshold for some time, and then suddenly cross it. Such events are not indicated by any trend in the request rates and therefore cannot be predicted, causing our prediction system to activate the server infrastructure only after a flash crowd actually starts.

However, while the problem with bursts cannot be recognized until after a flash crowd has started, our prediction system can easily detect oscillations around the capacity threshold. It can then decide to activate the server infrastructure preventingly even though the actual capacity threshold has not been crossed yet. We return to this issue below, when discussing the details of flash crowd prediction.

4.2. Architecture

In order to design a reliable prediction system that outputs activation recommendations it is necessary to have both the detection and the prediction algorithms working in parallel. The reasons for that are twofold. First, each of the two algorithms have different goals and performs better under certain traffic conditions. When the flash crowd is already in progress, the sensitive detection algorithm can be more of an advantage. The prediction algorithm, in turn, is responsible for identifying such trends that may actually lead to a flash crowd. Second, every prediction issued by the prediction algorithm should be verified by the detection algorithm. Thus, if an alarm is actually false, the system is able to identify such a situation and return to the normal operation mode. We therefore implemented the prediction system in a component manner with the detection and prediction algorithms as its core components.

Figure 4.2 depicts the modular architecture of our prediction system. The system input is first pre-processed by a component called aggregator, which groups fine-grain request rate samples into bigger portions to facilitate their analysis. Then, the aggregated samples are passed in parallel to two other components. The first of them, called a detector, implements the formal flash crowd definition to detect flash crowds when they are already in progress. The second component, called a predictor, implements the linear regression algorithm to analyse the trends in the received samples. The output of both these components is passed to the last component, called a merger, which makes the final decision about activating or de-activating the system infrastructure. The following sections discuss each of the four components in detail.

4.2.1. Aggregator

Depending on the system infrastructure where the predictor runs, the request rate samples received by the prediction system might be very fine-grained such as one sample per second. Meanwhile, both the formal flash crowd definition implemented by the detector and the linear regression implemented by the predictor require relatively coarse-grained samples. The detector uses coarse-grained samples to verify whether the average request rate exceeds the capacity threshold. The predictor, in turn, needs coarse-grain samples to reliably identify trends in the request rate.

To meet the requirements of the detector and the predictor, the prediction system first aggregates the request rate samples in the aggregator. The aggregation is performed in two steps. First, the data is aggregated into samples of ρ_d seconds, which corresponds to the resampling interval of the detector. At this stage, the samples are ready to be passed to the detector.

However, as shown in Chapter 5, reliably detecting trends in the request rate mandates that the samples are aggregated even further. To this end, the aggregator performs the second-step aggregation, in which the samples are grouped into larger samples of ρ_p seconds. Note that, as a side effect, using more coarse-grained leads enables the prediction system to work faster, as it then has fewer samples to process.

4.2.2. Detector

The request rate samples produced during the first-step aggregation are passed to the detector, which implements the formal flash crowd definition in order to detect flash crowds and bursts. For each aggregated sample, the detector outputs a boolean value denoting whether the system is already in a flash crowd according to the formal flash crowd definition.

The operation of the detector depends on three parameters. We now provide a brief description of them, and present their detailed study in Chapter 5.

The first parameter of the detector is the window size W_d, which denotes the size of a window over which the average request rate is computed. In general, the size of the window affects sensitivity of the detector. Using large windows might therefore lead to ignoring many bursts.

The choice of a window size influences the reaction time of the detector. While detecting a flash crowd, the average request rate of samples is computed over the time of W_d seconds. The detector therefore obtains information about a flash crowd after $\frac{W_d}{2}$ seconds on average, which might already be too long when the window size is very large. On the other hand, using small windows causes the detector to recognize even short-lasting overloads as flash crowds. Note that the window size W_d should be a multiple of the resampling parameter ρ_d used by the aggregator so that the window always contains the same number of samples.

Another parameter of the detector is the low watermark H_d. As opposite to the high watermark, which defines the system capacity and therefore cannot be changed, the low watermark causes the detector to unify closely situated peaks, thereby reducing the number of necessary adaptations. In general, a too low H_d would result in ever-lasting flash crowd, whereas too high values of H_d reduce the detector's ability to connect peaks. However, choosing the right value of H_d is not easy, as it strongly depends on the burstiness of the request rate itself.

The last parameter of the detector is W_e, which is the interval during which the request rate should remain in the white zone before the flash crowd ends. Recall that this interval enables the detector to ignore "negative bursts", which are short-lasting decreases in the request rate during a flash crowd. In practice, the length of that interval corresponds to the time when the infrastructure of mirror servers may remain allocated even though the flash crowd is over. This time typically depends on the cost of utilizing such an infrastructure.

Figure 4.3: Refined interface of the linear regression predictor.

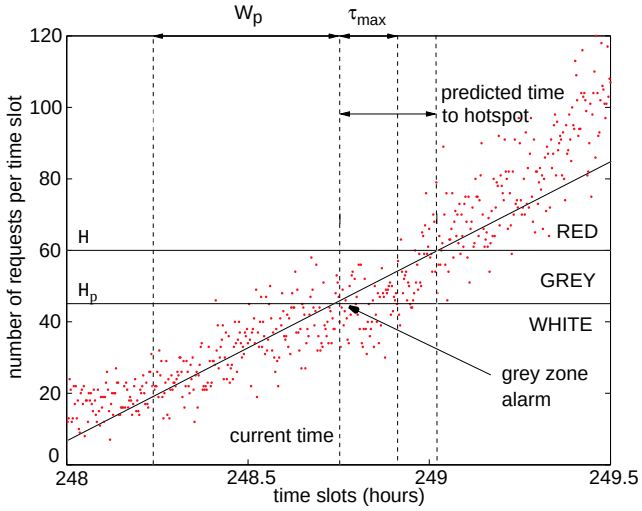


Figure 4.4: Grey zone alarm in predictor.

4.2.3. Predictor

The core of our prediction system is the predictor, which applies the linear regression algorithm to the stream of request rate samples received from the aggregator. In principle, the linear regression works exactly as described in Section 2.3. However, instead of returning a pair of (time, prediction) where "prediction" denotes the output of linear regression, our predictor uses a more sophisticated interface to convey more information to the merger.

More specifically, while analysing trends in the request rate, our predictor can not only predict future request rates, but also identify if the present request rate approaches a dangerously high level. This capability enables the predictor to inform the merger about the situations when trend analysis makes no sense, as the request rate remains so close to the capacity threshold that it may be exceeded at any moment. Such information is extremely valuable to the merger, as it then might decide to preventingly activate the infrastructure of mirror servers. Thus, we are able to solve the problem of an oscillating traffic request rate, described at the beginning of this chapter.

The predictor identifies the oscillations around the capacity threshold by means of the watermarking technique. Similar as with the detector, it defines three zones for the current request rate: white, grey, and red. The difference, however, is that the grey zone is much thinner than in the detector, as the predictor sets its low watermark at around 90% of the capacity threshold. In that case, the sole fact of entering the grey zone can be a good reason to activate the infrastructure of mirror servers, as the request rate is already very close to the capacity threshold (see Figure 4.4). Another difference is that the predictor determines the zone using linear regression and not load averaging, as it is interested in the current situation rather than in average situation over some recent past.

The refined predictor interface can therefore be defined as follows (see Figure 4.3):

Definition 4.1 *We denote the result of the prediction algorithm as a triplet* $(t_1, zone, t_2)$. t_1 *is time at*

which the prediction was given. $Zone$ corresponds to the load zone to which belongs the request rate at time t_1. This is the result of the linear regression with $\tau_{max} = 0$. t_2 is time for which the prediction was issued, $t_2 \in [0, \tau_{max}] \cup \infty$. In case the flash crowd is predicted to occur too far away in the future, then $t_2 = \infty$. If the flash crowd has already started or has just been detected, then $t_2 = 0$. Otherwise t_2 is time at which the linear regression line crosses the flash crowd level H.

In order to better understand the behaviour of the predictor and how it can be interpreted during the postprocessing phase, we discuss the actual semantics of the returned triplets. Being in the white zone $(t_1, white, \infty)$ is the normal situation, in which the request rate is much below the capacity threshold. However, if the request rate starts to grow fast enough to exceed the high watermark within a short amount of time, the prediction algorithm returns a warning and a predicted time to flash crowd $(t_1, white, t_2)$.

Being in the grey zone, in turn, is an indicator of a dangerously high request rate. The possible predictor outputs are $(t_1, grey, \infty)$ and $(t_1, grey, t_2)$. Although t_2 might still be provided, it should only be treated as an additional piece of information. Also, the value of t_2 might be inaccurate, as the linear regression algorithm works based on a relatively long window of samples compared to the very small size of the grey zone.

Finally, while in the red zone, the predictor does not need to predict any flash crowd as it is already in progress. It can therefore return only the information about the current zone: $(t_1, red, 0)$.

Similar to the detector, the operation of the predictor depends on a number of parameters. The influence of the prediction window W_p and the maximum advance notice τ_{max} on the predictor performance has already been discussed in Section 2.3. In addition to that, the prediction window W_p is usually a few times larger than W_d used by the detector. This is because W_d determines the magnitude of flash crowds detected by the prediction system, and must by nature be short to ensure the detector's sensitivity. W_p, in turn, is used to analyze longer-lasting trends, which can be reliably identified only when W_p is large enough.

The general parameters defining a flash crowd, including the watermarks and the W_e interval, apply to the predictor as well. Similar as in the detector, the high watermark and the W_e interval reflect the capacity and cost of the system. The low watermark H_p, however, defines the beginning of the predictor's grey zone and as such should be carefully chosen. In most cases, it is better to set it much higher than the low watermark H_d used by the detector. We return to this issue when evaluating our prediction system below.

4.2.4. Merger

Both the detector and the predictor pass their output to the merger, which makes the final decision about whether the infrastructure of mirror servers should be activated or de-activated. We discuss the decision-making process according to the possible outcomes it might have, starting from the normal operation without any flash crowds, through the detection and handling of a flash crowd, and until that flash crowd ends.

The operation of the merger implements the state machine depicted in Figure 4.5. During the normal operation, the high sensitivity of the detector might cause it to occassionally return a false alarm when it detects a burst. Therefore, the merger relies more on the predictor at this stage and waits until an alarm is issued by the predictor (state **NORMAL**).

Given that the predictor identifies trends in the request rate, it is supposed to discover whenever the request rate is growing significantly, and to issue a warning about an upcoming flash crowd before the detector does. The predictor issues two types of warnings. First, it might detect an upcoming flash crowd based on the trend in the request rate while still in the white zone. Second, it might detect that the current request rate has entered the grey (or even red) zone. Both these situations cause the

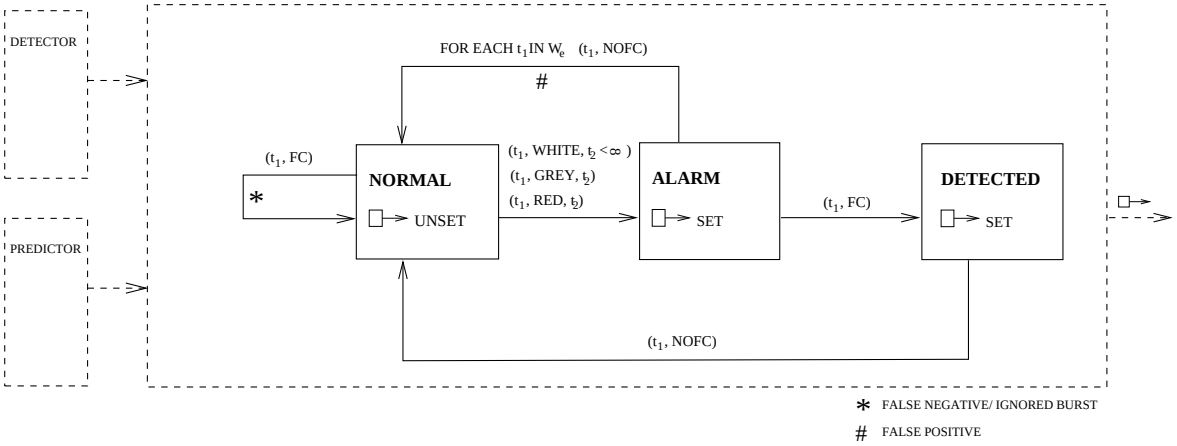


Figure 4.5: Merging state machine.

merger to switch to the **ALARM** state, in which the prediction system recommends to activate the infrastructure of mirror servers.

While in the **ALARM** state, the merger is waiting for the flash crowd to be confirmed by the detector, which shall cause the state to be changed to **DETECTED**. However, if no confirmation arrives within W_e, the merger returns to the **NORMAL** state, and recommends to de-activate the infrastructure. This is consistent with the nature of the W_e parameter, which defines the maximum time for which the infrastructure can remain active and unused.

The **DETECTED** state corresponds to the situation when the flash crowd has already been confirmed by the detector. The merger keeps recommending infrastructure activation initiated in the **ALARM** state, and waits until the flash crowd is over, which is reported by the detector. Once that happens, the merger switches its state back to **NORMAL**, and recommends to de-activate the infrastructure.

An important observation is that the infrastructure is activated at time t_1, as soon as an alarm is raised, even though the predictor might indicate that the time remaining until the start of the flash crowd is t_2. There are two reasons for that. First, recall that $t_2 \in [0, \tau_{max}]$ where τ_{max} denotes the maximum time in the future for which prediction can be issued. As we have already discussed, predictions based on a too large τ_{max} tend to be unreliable. The value of τ_{max} should therefore be relatively small, especially compared to the W_e interval. This causes the cost of infrastructures activated at t_1 and t_2 to be very similar. Second, the growing intensity of a flash crowd might cause the request rate to reach the high watermark faster than initially expected, justifying earlier infrastructure activation. As a consequence, the potential benefits of such an early activation far outweigh the possible losses.

Another observation is that the merger could be a perfect place to distinguish between flash crowds and bursts, thereby solving the first problem we mentioned at the beginning of this chapter. In principle, the merger could avoid activating the infrastructure during bursts by first confirming that the detected overload is a real flash crowd. To this end, it would wait for the minimum flash crowd duration time τ_{fc} before triggering the infrastructure activation. However, such an approach would also cause the prediction system to react too late to *all* the flash crowds, as each of them would first have to be confirmed. We therefore do not use the τ_{fc} parameter in our flash crowd prediction strategy, but only in the evaluation process as discussed in the next chapter.

Chapter 5

Evaluation

The goal of a good prediction system is to output accurate warnings of upcoming flash crowds. This, however, is not an easy task due to certain traffic properties. As a result, the prediction system may output warnings too late or too early when compared to the real flash crowd duration. In order to compare the quality of different prediction systems with respect to the number of false positive and false negative alarms they return, we first precisely define these properties. We then present the traces that will serve as example data sets for our prediction systems. Finally, we present results of different prediction system runs on these traces so that we can give guidelines on how to find optimal values of parameters for the prediction system.

5.1. Evaluation Settings

The detector actually defines what a real flash crowd for a given trace is, thereby allowing to count both false positives and false negatives. Therefore, the main idea for evaluating a given flash crowd prediction system is to compare the outputs of the prediction system and that of the detector and count the number of occurrences where they diverge. Importantly, we do not count a penalty for each time slot separately, but treat an undetected flash crowd or an unnecessary alarm as one error. Thus, the penalty corresponds to the number of events that are correctly/incorrectly detected and not to the number of slots in a given run, thereby avoiding correlation with the resampling interval of the detector during the evaluation stage.

More precisely, we compare the output streams of the detector and prediction system, as shown in Figure 5.1. The detector for each time slot returns a binary value “FC” or “NOFC”, which denote if there is a flash crowd or not. As described in Chapter 3 the “FC” samples form consecutive sequences of length at least W_e. We denote the start of such sequence by $t_{startFC}$ and its end by t_{endFC}.

The prediction system, in turn, outputs “SET” and “UNSET” recommendations for each time slot following the decision strategy presented in the previous chapter. The “SET” recommendations also form consecutive sequences which either last precisely W_e seconds (in case of a false alarm) or as long as the detector issues “FC” values (other cases). Similarly, we denote by $t_{startSET}$ the start of such sequence and by t_{endSET} its end.

5.1.1. False Positive Prediction

A false positive prediction depicts an adaptation that takes place too early before a real flash crowd to allow for efficient infrastructure usage (see Figure 5.2). We use the W_e parameter to denote the maximal interval before a flash crowd in which the adaptation can legitimately start. Note that this

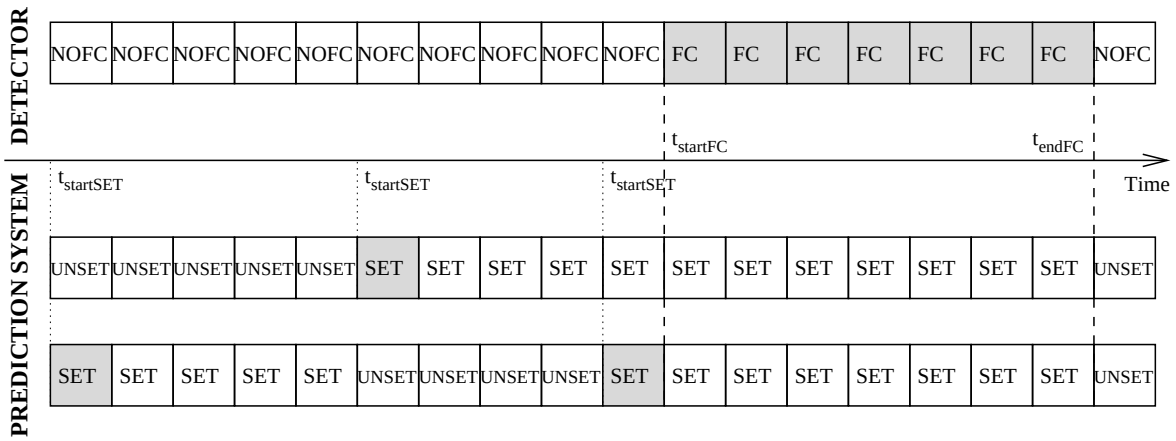


Figure 5.1: Detector and prediction system output comparison. Slots between $t_{startFC}$ and t_{endFC} are grey. Also slots $t_{startSET}$ are marked as grey.

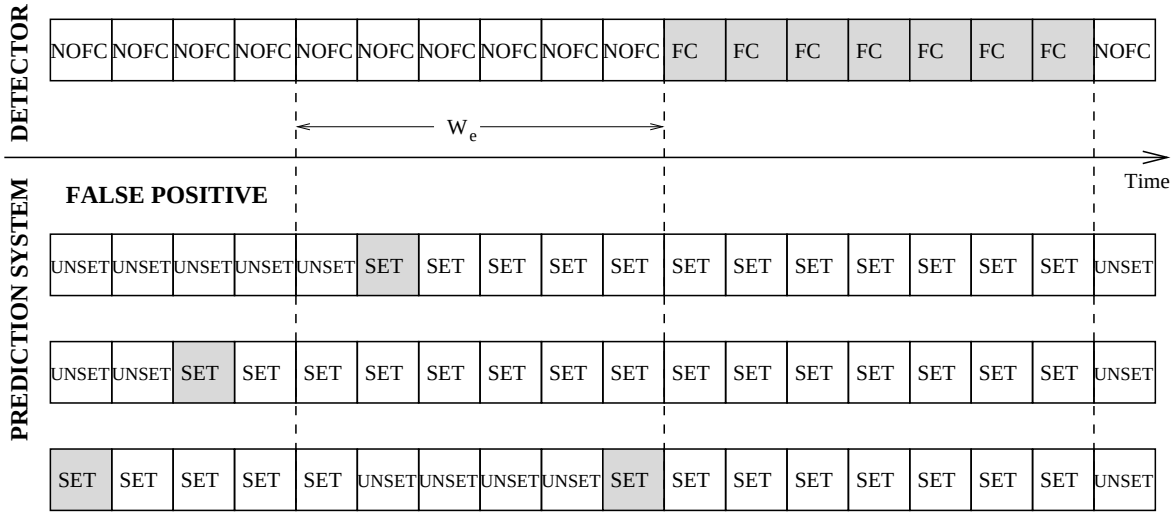


Figure 5.2: Detector and prediction system output comparison: false positives.

is consistent with the nature of W_e parameter, as it defines the time for which the infrastructure can remain active and unused. The false positive prediction is therefore as follows:

Definition 5.1 *A false positive prediction occurs for an adaptation starting at time $t_{startSET}$ when $t_{startSET} < t_{startFC} - W_e$.*

A special type of false positives are adaptations during which a short-lived overload, a burst, occurs. Bursts differ from flash crowds in such a way that they last shorter, that is less than τ_{fc}. Recall that we presented such a prediction system that tries to avoid adapting to bursts. However, bursts can look very similar to flash crowds in their early stages and therefore the prediction system might recommend an adaptation anyway. Such an adaptation is not as efficient in terms of infrastructure usage as a flash crowd adaptation, but it could still have some sense, depending on the user preferences. Therefore we introduce in the evaluation procedure the notion of a false positive adaptation during a burst, a *false burst* in short, whose definition is the following:

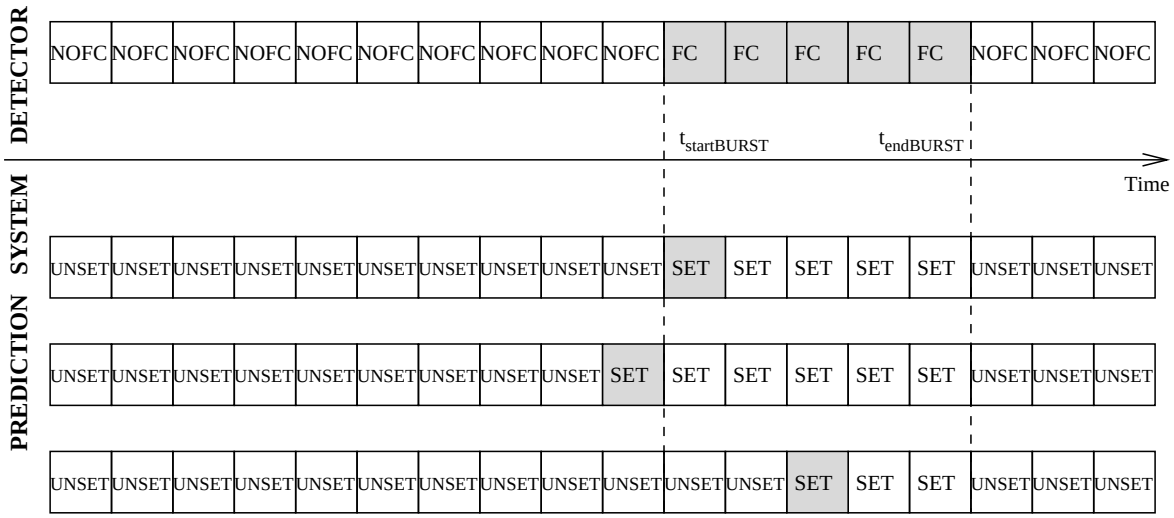


Figure 5.3: Detector and prediction system output comparison: false bursts.

Definition 5.2 *A false burst occurs for an adaptation starting at time $t_{startSET}$ when $t_{startBURST} \in [t_{startSET}, t_{endSET}]$ or $t_{startSET} \in [t_{startBURST}, t_{endBURST}]$ where $t_{startBURST}$ and $t_{endBURST}$ denote the start and end of a burst respectively.*

Possible situations in which an adaptation could be classified as a false burst are depicted in Figure 5.3.

5.1.2. False Negative Prediction

A false negative prediction occurs when the prediction system recommends no adaptation at all for a given flash crowd, or this adaptation starts too late. Figure 5.4 depicts possible false negative predictions, where the adaptation is supposed to start within δ_{neg} seconds before a flash crowd (if it is not in progress due to an earlier alarm already). The definition of a false negative prediction therefore states:

Definition 5.3 *A false negative prediction occurs for a flash crowd starting at time $t_{startFC}$ when no alarm is raised between $t_{startFC} - \delta_{neg}$ and $t_{startFC}$.*

5.1.3. Summary

We can therefore summarize the recommendations based on where they start and where the actual flash crowd begins as presented in Table 5.1. Notice that the “no penalty” period is the time between a false positive and a true positive. During this time we are actually indifferent whether the prediction system recommends us to activate the infrastructure or to remain in the normal operation mode.

5.1.4. Settings

In the following sections we evaluate our prediction system with the following parameter settings. Our goal is to predict flash crowds that last at least 10 minutes, which means that $\tau_{fc} = 600$ seconds. We expect the prediction system to recommend an adaptation within 150 seconds before a flash crowd

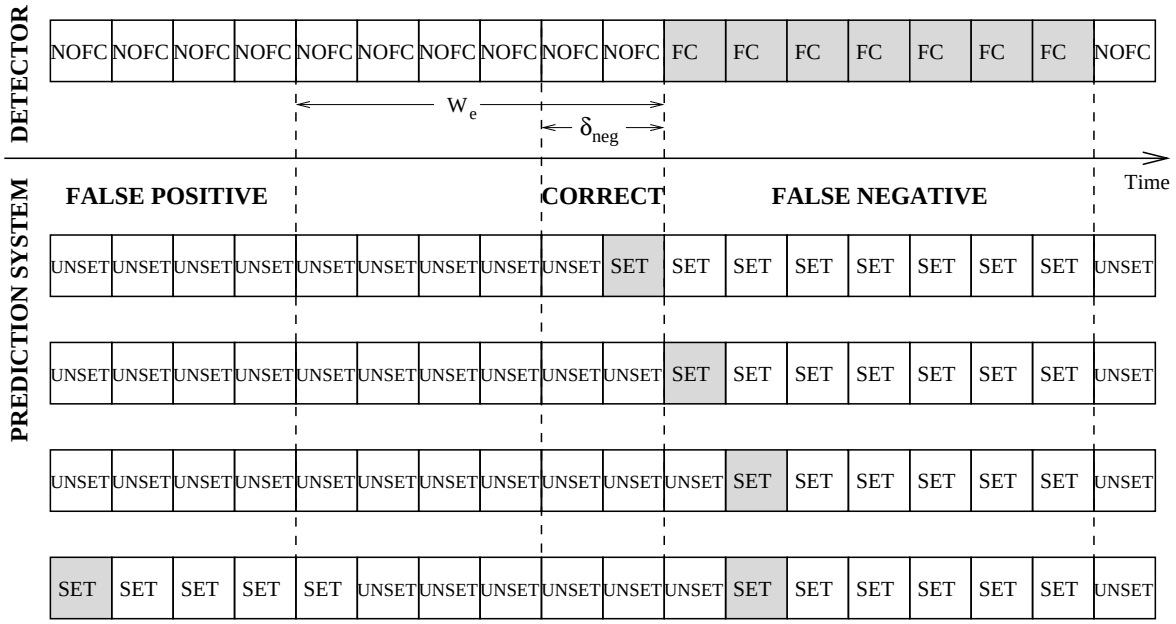
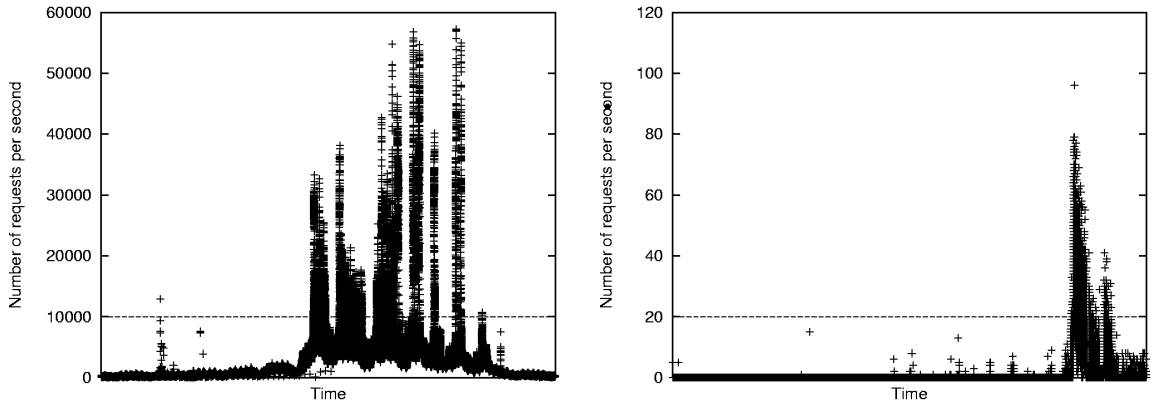


Figure 5.4: Detector and prediction system output comparison: false negatives.

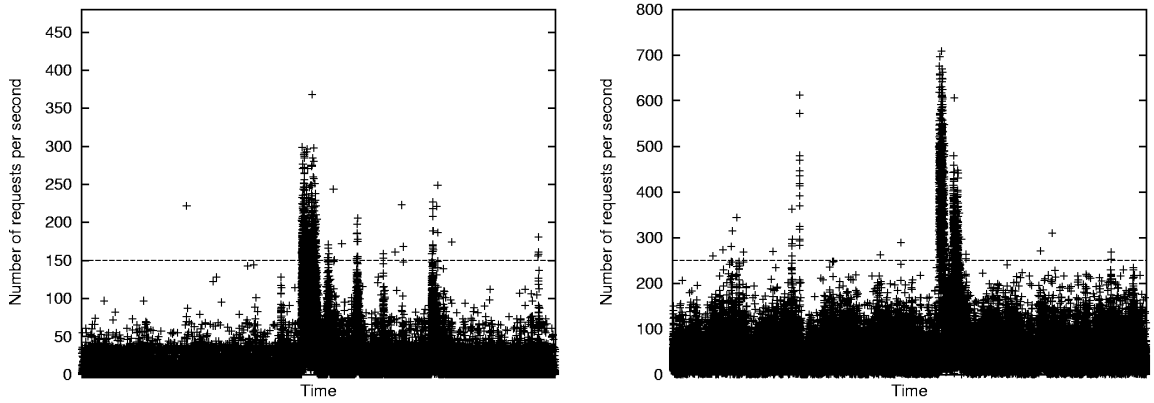
Start time	**Description**
$t_{startSET} < t_{startFC} - W_e$	False positive; wasted resources.
$t_{startFC} - W_e \leq t_{startSET} < t_{startFC} - \delta_{neg}$	No penalty; resources used at the end of W_e period.
$t_{startFC} - \delta_{neg} \leq t_{startSET} < t_{startFC}$	True positive; most efficient resources usage.
$t_{startFC} \leq t_{startSET}$	False negative; system unprepared.

Table 5.1: Classification of possible adaptation recommendations, which start at time $t_{startSET}$.



(a) Worldcup: 87 days ($H = 10000$ rps)

(b) Minix: 119 days ($H = 20$ rps)



(c) AST: 70 days ($H = 150$ rps)

(d) Fractal: 60 days ($H = 250$ rps)

Figure 5.5: Traces ($\rho_d = 15$ s).

starts ($\delta_{neg} = 150$). However, the adaptation can start earlier, but at most 30 minutes, which is the value of the W_e interval corresponding to the time the infrastructure can remain active and unused.

We decided to apply equal penalties to false negative and false positive predictions (set to 1 penalty unit), as they both should be avoided. The cost of a false burst, in turn, is set lower (0.5 units). The goal of the prediction system is to minimize the number of errors during a given run.

5.2. Traces

We evaluate our predictor on four traffic load traces, which have very different properties. We describe them briefly here so that we can develop intuition of what type of prediction errors might be expected.

Figure 5.5 presents the four traces aggregated by such a ρ_d that is used during the experiments (servers capacity is expressed in requests per second, rps).

The first trace is the Worldcup trace from the Soccer World Cup in 1998 [5]. There are several easily distinguishable flash crowds, which typically last several hours. Their periodic occurrence corresponds to the times when matches are played. In addition, there are up to two bursts (depending on the detector parameters), at the beginning and at the end of the trace. The capacity of the server is fairly large, what might be the reason for the flash crowds to be so noticeable and long.

The second trace is more bursty, coming from a smaller-sized server hosting the `minix3.org` Web site [14]. This is a classical example of a flash crowd trace, in which the flash crowd is preceded

Trace	Start Date	Duration (days)	H [rps]
Worldcup	30 Apr 1998	87	10000
Minix	24 Oct 2005	119	20
AST	18 Apr 2004	70	150
Fractal	31 Dec 2003	60	250

Table 5.2: Traces summary (rps denotes requests per second).

by a period of very low request rates. At some point, the Internet community becomes interested in the content hosted on the server, in this case the new release of the Minix operating system. This results in a flash crowd, which after a few days is followed by a decrease in the request rate to a moderate level.

Another bursty trace is the AST trace, which shows the popularity of Prof. Andrew Tanenbaum's home page during the course of his scientific debate with Ken Brown [13]. The largest flash crowd appears at the beginning of the trace after a period of a rather low request rate. This peak is followed by a few others, but of a smaller magnitude than the first one. Outside the flash crowd periods the request rate remains at about 40% of the servers capacity.

The last trace used during the evaluation is the Fractal trace from an Australian Web server hosting a page about fractals [7]. The trace is also very bursty with the request rate usually reaching 80% of the servers capacity. There are several sudden peaks in the request rate which are not indicated by any previous traffic behaviour, what makes the predictions a rather hard task. In addition, the typical level reached by the request rate is distinctly higher than that of the AST, for example. As a consequence, the prediction system may issue many warnings related to the grey zone entries.

We should note that the capacity thresholds were set arbitrarily. For example we set it lower in the Fractal trace, in comparison to the AST trace, so that we can check how our prediction system behaves on a similarly bursty trace, but under much harder conditions which actually resemble a permanently overloaded server. Table 5.2 summarizes the basic information about the four traces.

5.3. Experiments

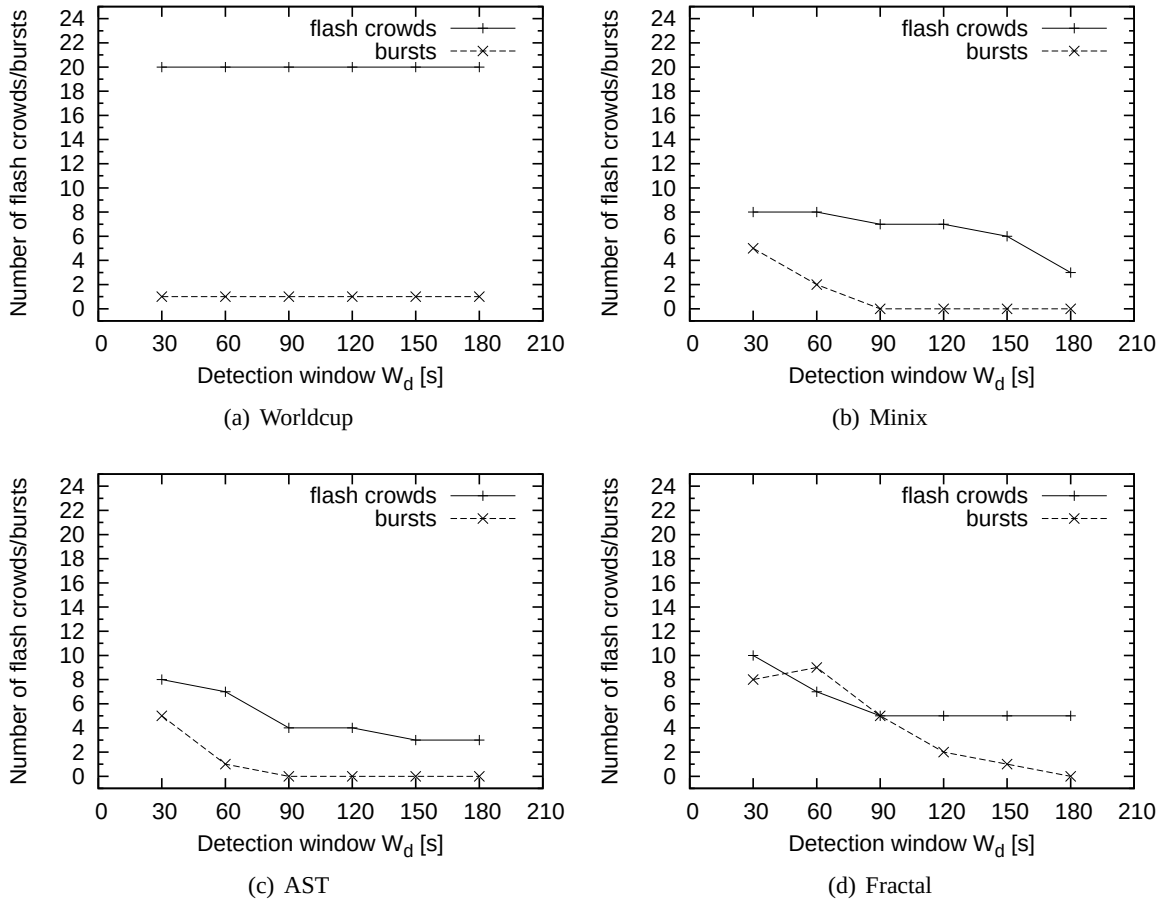
In order to compare different prediction systems we have to define the magnitude of flash crowds that are to be predicted. Therefore we first choose settings for the detector, whose output will serve as a reference point for all prediction systems run on this trace. Consequently, the structure of this section is as follows. First, we discuss parameters that influence detection. Then, we present short characteristics of our prediction system and choose predictor settings that behave best for the given traces.

5.3.1. Detection

This section discusses how to select parameters that influence the detection process. They either depend on human judgement or are coupled with the handling infrastructure deployed in the system.

W_e Interval

The W_e parameter is chosen with respect to the infrastructure cost, independently from the detection process. It influences the detection in such a way that it unifies closely situated peaks, thereby causing flash crowds to last longer.



(a) Worldcup (b) Minix

(c) AST (d) Fractal

Figure 5.6: Different detection windows.

Resampling Interval ρ_d

In all our experiments the resampling interval ρ_d is set to 15 seconds because of two reasons. First, it is very close to the resampling interval used in the initial paper on flash crowd prediction where it was set to 10 seconds (for details see Section 2.3). Second, detectors that differ little in setting of the ρ_d parameter usually output almost identical flash crowd sequences. However, larger values of ρ_d speed up the computation.

Detection Window W_d

The size for the detection window W_d greatly influences the magnitude of the flash crowds that are detected. Figure 5.6 illustrates the correlation between the size of the detection window and the number of detected flash crowds and bursts. In the experiments $\rho_d = 15$ s and $H_d = 0.7 * H$ rps (the low watermark is discussed in detail below).

Recall that small window sizes cause the detector to be very reactive, whereas larger windows allow for more stability. This can be easily seen when comparing the detection results in Figure 5.6 for $W_d = 30$ and $W_d = 180$ seconds. A detector with such a small window size as 30 seconds recognizes an overload very fast as either a flash crowd or a burst. A detector with a very large window size like 180 seconds, in turn, obtains information about a flash crowd only after 90 seconds. An exception here is the very regular Worldcup trace, whose results are all independent from the window size.

Trace	H [rps]	Detector			Number of Flash Crowds	Number of Bursts
		H_d [rps]	ρ_d [s]	W_d [s]		
Worldcup	10000	0.6*H	15	90	19	1
Minix	20	0.6*H	15	90	5	0
AST	150	0.6*H	15	90	4	0
Fractal	250	0.6*H	15	120	4	2

Table 5.3: Traces summary: detector's stage.

Defining where the flash crowd actually starts and ends is quite subjective. Therefore we cannot give here an ultimate answer with one detection window size suitable for each and every trace. However, based on the observations of Figure 5.6 we propose to distinguish between "small" and "large" window sizes. As "small" we define windows of less than 60 seconds, whereas "large" windows are those that have at least 150 seconds. Both window types have their advantages and drawbacks as described above. We therefore believe that it is reasonable to choose a window that is between the two extremes.

Closer investigation of Figure 5.6 shows that for the Worldcup, Minix and AST traces the number of flash crowds remains fairly stable for windows of 90 seconds and larger. Importantly, bursts for these window sizes are eliminated. The Fractal trace, in turn, produces a distinctly larger number of bursts in comparison to the number of flash crowds. For the larger window sizes the number of bursts decreases, but they are hard to eliminate. The reason for that is probably the fact that the Fractal trace simulates an overloaded server trace, where load quite often increases above the high watermark. As a , we use $W_d = 90$ s during the tests of our prediction system for the Worldcup, Minix and AST traces. For the more bursty Fractal trace, it is better to choose a larger window $W_d = 120$ s, as this might reduce the number of bursts.

Low Watermark H_d

The final parameter we need to set in the detector is the low watermark H_d, which connects closely situated request rate peaks. As depicted in Figure 5.7 this parameter typically helps to reduce the number of detected flash crowds. However, setting this parameter lower also influences the duration of both flash crowds and bursts. As a consequence, it not only helps to merge several flash crowds into one, but produces new flash crowds from peaks previously recognized as bursts. Such a situation occurs, for example, when analysing the Fractal trace. The second burst in the Worldcup trace has also turned into a flash crowd after the low watermark crossed the 0.8*H value. In general, we advise to choose this parameter between 0.6 and 0.7 of the capacity server. In the experiments presented below we set the low watermark at 0.6*H rps.

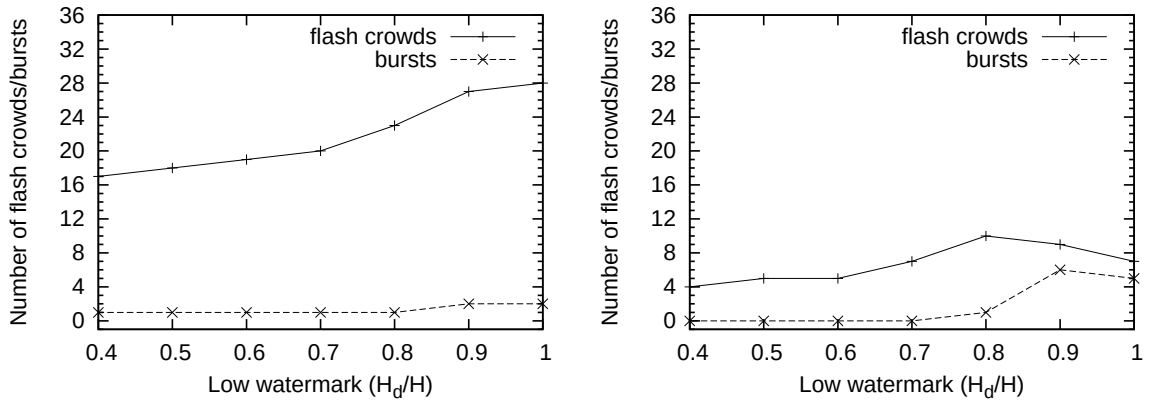
Summary

Table 5.3 summarizes the current evaluation settings. In the next section all prediction systems are evaluated against detectors with such settings.

5.3.2. Prediction

This section presents how the prediction system, in particular the linear regression predictor, works with the detector described above.

There are four parameters that need to be set. We test our prediction system against four values of the resampling interval ρ_p: 15, 30, 45 and 60 seconds. For each of these values we try to find such



(a) Worldcup ($W_d = 90$ s)

(b) Minix ($W_d = 90$ s)

(c) AST ($W_d = 90$ s)

(d) Fractal ($W_d = 120$ s)

Figure 5.7: Different low watermark settings.

combination of the prediction window W_p (values from the range between 150 and 720 seconds) and the predictors low watermark H_p (values between 0.95*H and 0.80*H rps) that the number of errors is minimal. The parameter that remains constant throughout the experiments is the maximal advance notice τ_{max}, which is set to 5 minutes.

Figure 5.8 presents the overall results of our prediction system. Each subfigure shows the respective number of errors depending on the window size W_p and the resampling interval ρ_p. Results with two low watermark values are presented: in the left column are the results for $H_p = 0.95 * H$, whereas the right one shows the outputs for $H_p = 0.90 * H$. Thus, we can most easily compare the results for all the traces and find common patterns concerning the curve shapes. While analyzing Figure 5.8 it is important to compare the outputs of prediction system with respective numbers of flash crowds and bursts given in Table 5.3.

Before we describe all parameters separately, notice that the curves for different ρ_p values are all "U-shaped". In general, the values for which the curves reach their minimum are different for each trace. Therefore we should not expect to find a single best value for all of them. However, we can determine default values that will allow reasonable predictions for all studied traces.

Prediction Window W_p

While analyzing the presented data, it is clear that for most cases a window size lower than 300 seconds tends to generate a distinctly greater number of errors in comparison to larger windows. Note that smaller values cause the prediction system to issue many warnings, whereas larger values stabilize at such levels that practically no flash crowds are predicted on time. We therefore propose to set W_p between 420 and 600 seconds.

Recall that prediction window influences reactiveness of the linear regression predictor, as depicted in Figure 5.9. Prediction system with smaller windows generates many false positive alarms, whereas large windows have great impact on the number of false negatives issued. Selecting a larger prediction window also helps to ignore bursts, which is the consequence of reducing the overall number of false positives. This general rule is confirmed by the U-shape of the respective curves from Figure 5.8, as false positives and false negatives sum to the final number of errors.

Low Watermark H_p

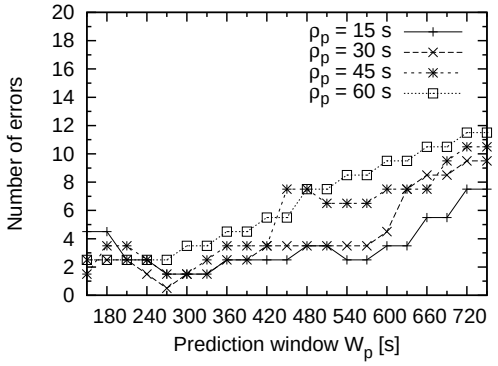
As can be observed in Figure 5.8 the minimum for most of the curves moves towards larger W_p values when the low watermark H_p is set smaller. For the already chosen safe W_p settings there is, however, no clear difference when the results for 0.95*H and 0.90*H are compared. We therefore set it simply to 0.90*H rps. The only exception is the Worldcup trace where the minimum is reached for smaller window sizes and optimal settings can be found for $H_p = 0.95 * H$ rps.

Selecting the right value for the low watermark H_p is again a trade-off between the number of false positives and false negatives. Recall that the low watermark reduces the number of false negatives, as the prediction alarms are issued preventingly at the time the predictor reaches H_p. Setting the low watermark too low, however, might greatly increase the number of false positives, as illustrated in Figure 5.10.

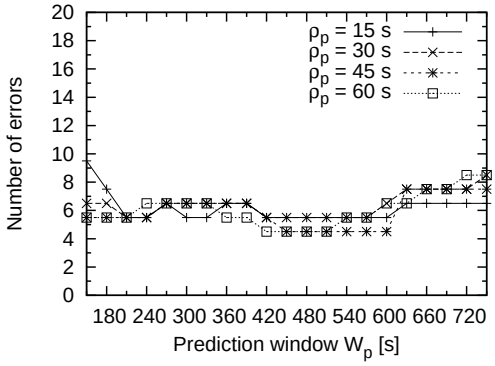
Note how distinctly higher this parameter is set in comparison to the low watermark used in the detector (typically $H_d = 0.6 * H$ or $H_d = 0.7 * H$ rps).

Resampling interval ρ_p

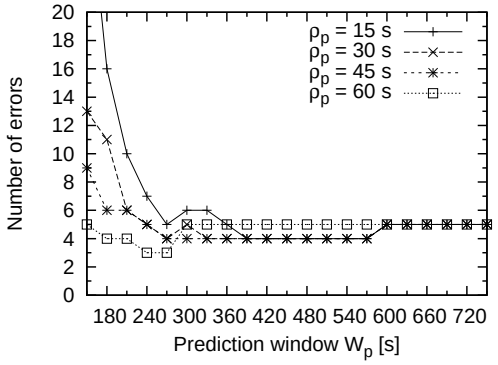
Figure 5.8 suggests that for W_p between 420 and 600 seconds and $H_p = 0.90 * H$ rps it is safe to select the ρ_p parameter at 45 seconds.



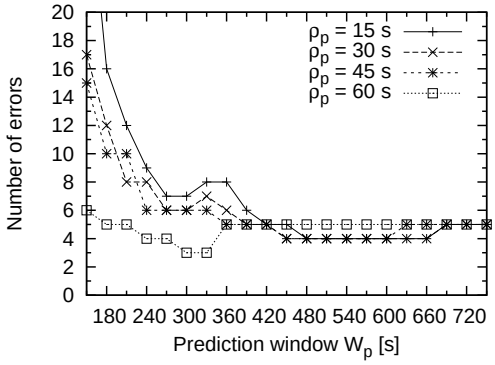
(a) Worldcup $H_p = 0.95 * H$ rps



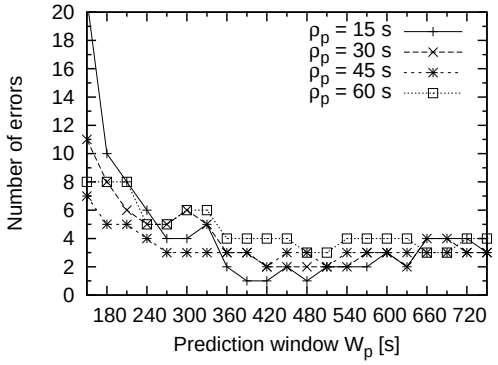
(b) Worldcup $H_p = 0.90 * H$ rps



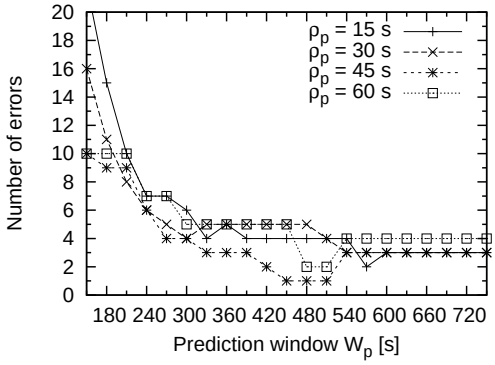
(c) Minix $H_p = 0.95 * H$ rps



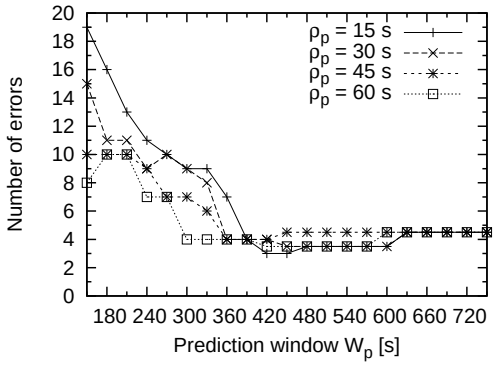
(d) Minix $H_p = 0.90 * H$ rps



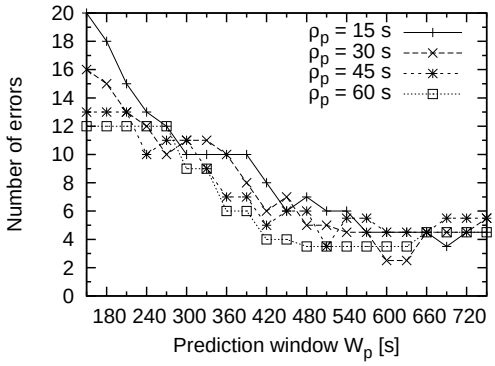
(e) AST $H_p = 0.95 * H$ rps



(f) AST $H_p = 0.90 * H$ rps



(g) Fractal $H_p = 0.95 * H$ rps



(h) Fractal $H_p = 0.90 * H$ rps

Figure 5.8: Prediction results.

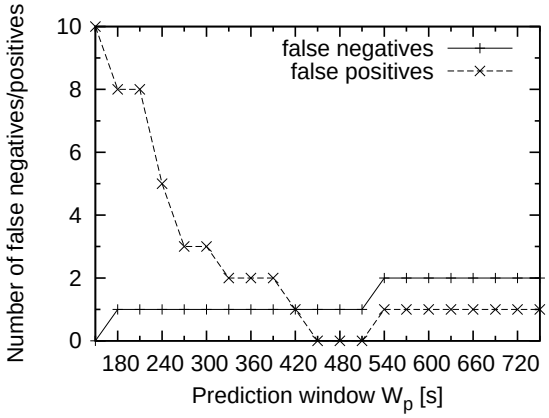


Figure 5.9: Correlation between the window size W_p and the number of false negatives and false positives (AST trace: $H_p = 0.9 * H$ rps, $\rho_p = 45$ s).

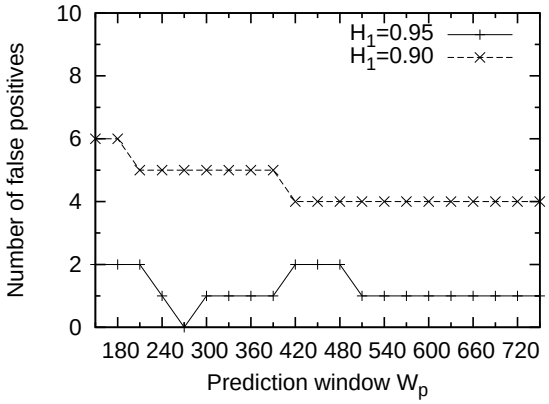


Figure 5.10: Correlation between setting the low watermark and the respective number of false positive alarms (Worldcup trace: $\rho_p = 30$ s).

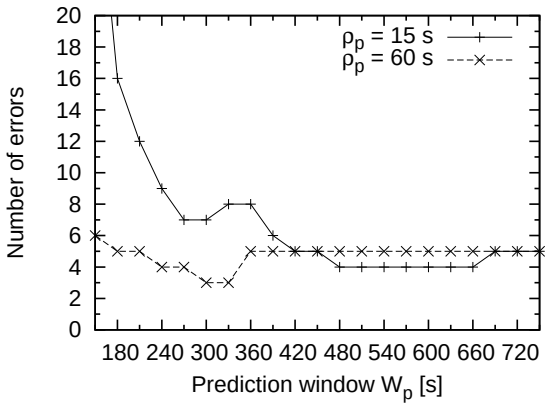


Figure 5.11: Correlation between ρ_p and the number of generated errors (Minix trace: $H_p = 0.9 * H$ rps).

The ρ_p parameter is mainly responsible for smoothing the data, thereby allowing the predictor to more easily discover trends in the request rate. The burstiness of the trace has a great impact on setting ρ_p, as the more bursty traces need to be aggregated with larger ρ_p values. On the other hand, if we expect the prediction system to be rather reactive it might be better to choose a smaller value for ρ_p. In general, the results of prediction system with larger ρ_p are more stable (the U-shape of the curves flattens; the W_p is less important), but it might be hard to find the optimal prediction values for a given trace. Figure 5.11 illustrates more clearly this dependency, already noticeable in Figure 5.8.

We treat the above remarks as guidelines, since we have not defined how to measure the trace burstiness precisely. One of the possible solutions to determine this property could be the autocorrelation function. However, we leave this issue for future work.

Maximum Advance Notice τ_{max}

In general, it turns out that the influence of τ_{max} on the prediction process is negligible (we compared $\tau_{max} = 5$ and $\tau_{max} = 10$ minutes). As one could expect, the prediction system with a larger τ_{max} generates more false positive alarms, but only when the prediction window is extremely small. We explain this loss of influence by the fact that the majority of the prediction alarms is issued due to grey zone crossing.

Discussion

Obviously, the prediction system with default parameters (as presented in Table 5.4) cannot be optimal for each and every trace. In order to improve its performance, the prediction settings can be tuned if additional trace characteristics are available. We therefore discuss the possible improvements based on the results obtained for the four example traces.

Worldcup: an example of a low bursty trace With the generally safe parameter settings, the prediction system clearly does not perform optimally. The reason is that the trace has very low burstiness. In order to improve the performance we need to change the safe values for all the parameters. First, recall the difference in number of false positives for the two low watermarks presented in Figure 5.10. It is clear that especially for the Worldcup trace it is better to set $H_p = 0.95 * H$. Second, note that smaller resampling interval ρ_p is more suitable for this trace (especially for $H_p = 0.95 * H$).

Trace	H [rps]	Detector			Predictor		
		H_d [rps]	ρ_d [s]	W_d [s]	H_p [rps]	ρ_p [s]	W_p [s]
Worldcup	10000	0.6*H	15	90	0.90*H	45	480
Minix	20	0.6*H	15	90	0.90*H	45	480
AST	150	0.6*H	15	90	0.90*H	45	480
Fractal	250	0.6*H	15	120	0.90*H	45	480

Table 5.4: Traces summary: safe parameter settings (rps denotes requests per second).

Third, notice that the overall trend of the different ρ_p curves has opposite properties to the generally observed. More precisely, the best performance is observed for the window sizes *smaller* than 360 seconds. We therefore advise the following settings: $H_p = 0.95 * H$ rps, $\rho_p = 30$ s and $W_p = 300$ s.

The reason for such changes from safe to optimal settings is the low burstiness of the Worldcup trace. Since the request rate growth is stable, the prediction system returns practically no false negative predictions. Therefore setting the low watermark too low can only be source for unnecessarily preventing alarms. Similarly, there is no need to reduce the predictor's sensitivity by using larger aggregation and larger prediction window.

Interestingly, it is impossible to find such parameter settings that would allow to ignore the burst from the beginning of the trace. This is a typical example of a burst, which is a random traffic fluctuation with an extreme request rate growth, well beyond the capacity threshold. Sooner or later such sudden traffic surge has to influence the trend of the request rate samples.

Minix: an example of a trace with sudden flash crowds The stability of the error curves for different prediction windows and resampling intervals allowed us to quickly select safe parameter settings for this trace. As can be observed in Figure 5.8 the number of errors with optimal settings could only be reduced by one (with larger resampling interval and smaller prediction window). Note, however, that even with the optimal parameters selected still three out of five flash crowds would not be predicted early enough.

In order to explain this issue, we compared the respective outputs of the detector and prediction system ($W_p = 300$ s, $H_p = 0.90 * H$ rps, $\rho_p = 60$ s). Surprisingly, the prediction system behaves on this trace as a detector rather than as a real predictor. We identified two reasons responsible for such situation. First, flash crowds in the Minix trace start fairly suddenly, as the average request rate computed by the detector crossed from the white into the red zone in only 90 seconds. Second, the upward trend of the samples before three flash crowds was not clear enough for the prediction system to raise an alarm on time. Figure 5.12 depicts two different flash crowds in this trace. The second flash crowd illustrates the problem, as here the majority of the samples during this flash crowd remains below the capacity threshold.

AST: an example of a typical trace The default settings of the prediction system are practically the best settings that could be selected for the AST trace. The request rate also increases quite suddenly before the four flash crowds (as compared for example to the Worldcup trace), but the prediction system is able to issue an alarm before the flash crowd starts. The only exception is the last flash crowd, where the average request rate exceeds the high watermark in two minutes (starting from 0). However, such cases might well be classified as "unpredictable".

Fractal: an example of a trace from an already overloaded server The final observation refers to the Fractal trace, by which we tried to simulate a prediction on a server that is already approaching its maximum capacity before the flash crowds start. Because of the constantly high request rate and

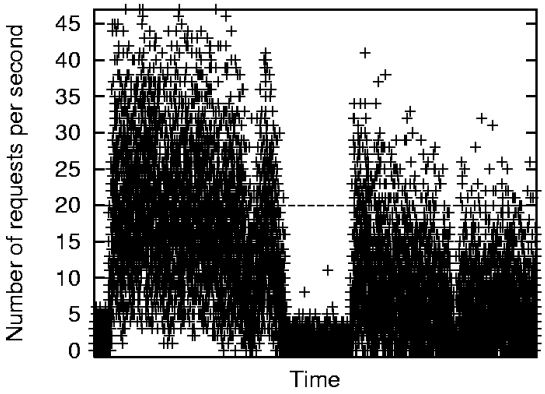


Figure 5.12: Early phase of two flash crowds in the Minix trace (duration: 60 hours).

many bursts identified, we already had to increase the size of the detection window. As can be seen in Figure 5.8, it is equally hard to find the optimal values for the predictor. In order to reduce the number of false positives, we could set the prediction window larger than for other traces and aggregate the samples even further. Increasing the low watermark value might seem a good idea as well, as the results for prediction system with $H_p = 0.95 * H$ are more stable. Note, however, that they stabilize at such level that practically no flash crowds are predicted on time (three out of four).

Unfortunately, the best our prediction system can do in such circumstances is to turn into a detector. In case of such a bursty trace, we believe that it is better to rely on a good detector that is able to ignore as many bursts as possible. Thus, the infrastructure would be allocated late, but only it is really necessary.

Chapter 6

Conclusions

In this thesis we have discussed current flash crowd handling techniques and argued that early adaptation is essential for effective dealing with flash crowds. In principle, timely adaptation is possible only when flash crowds are predicted. The problem of flash crowd prediction has been already addressed in a recent paper, which provided theoretical foundations by proposing to predict flash crowds using linear regression [6]. We have exploited this theory in our flash crowd prediction system.

The improvements over the original solution are twofold. First, we introduced a new flash crowd definition that allows for efficient resources usage. Our definition solved three main problems when identifying flash crowds. It detects the situations when the request rate oscillates around the Web site capacity, which results in more accurate flash crowd detection. Furthermore, it ignores negative bursts, thereby reducing the number of unnecessary infrastructure activations. Finally, it distinguishes between flash crowds and bursts so that the prediction system can precisely define when adaptation is really needed.

The second improvement is related to the prediction algorithm itself. We use the algorithm as presented in the paper, but we refine the predictor interface so that a special type of warnings can be issued. When the request rate oscillates just below the capacity threshold, these warnings are extremely useful, as in that case the linear regression algorithm is of little use. Another consequence of using such warnings is that our system does not trigger adaptations upon most bursts.

We have evaluated our prediction system by computing the number of errorenous decisions it produces based on a number of real-world flash crowd traces. In principle, these errors can be of two types. Either the prediction system recommends an unnecessary adaptation (false positive) or such a recommendation is issued too late (false negative). We have demonstrated that the performance of our system greatly depends on a number of configuration parameters, and that the system can be configured in such a way that it achieves reasonably good results on all the studied traces. In other words, the number of errors made by the prediction system with such settings is, although not optimal, rather low. Finally, we have shown that the system performance can be improved if additional trace characteristics are available.

We identify two main fields for exploitation in order to increase usability of our prediction system. First, trace classification with respect to its burstiness would allow to identify safe configuration parameters settings for each group of bursty traces separately, thereby improving performance of our prediction system. We believe that autocorrelation function could be used to that end.

Another field for exploitation we see in distributed environment, for example the prediction system could be deployed in a RaDaR-like architecture. We envision a solution where the prediction system runs on each server separately and sends reports about the current flash crowd situation to a special meta-prediction component deployed inside the replicator. The replicator can then decide how to change the redirection policy or whether to deploy new replicas.

Bibliography

[1] First Statement of the Food Agency in Belgium (AFSCA). http://www.influenza.be/fr/persberichten/2005-10-27_afsca_obligation_de_confinement.doc.

[2] Second Statement of the Food Agency in Belgium (AFSCA). http://www.influenza.be/fr/persberichten/2005-10-28_afsca_obligation_de_confinement.doc.

[3] ADLER, S. The Slashdot Effect: An Analysis of Three Internet Publications. http://ssadler.phy.bnl.gov/adler/SDE/SlashDotEffect.html, 1999.

[4] AKAMAI. http://www.akamai.com/.

[5] ARLITT, M., AND JIN, T. A Workload Characterization Study of the 1998 World Cup Web Site. *IEEE Network 14*, 3 (May 2000), 30–37.

[6] BARYSHNIKOV, Y., COFFMAN, E. G., PIERRE, G., RUBENSTEIN, D., SQUILLANTE, M., AND YIMWADSANA, T. Predictability of Web-Server Traffic Congestion. In *Proceedings of the Tenth IEEE International Workshop on Web Content Caching and Distribution* (September 2005), pp. 97–103.

[7] BOURKE, P. Googleblatted and SlashDotted, Feb. 2004. http://astronomy.swin.edu.au/~pbourke/fractals/quatjulia/google.html.

[8] FELBER, P., KALDEWEY, T., AND WEISS, S. Proactive Hot Spot Avoidance for Web Server Dependability. In *23rd International Symposium on Reliable Distributed Systems (SRDS 2004), 18-20 October 2004, Florianpolis, Brazil* (October 2004), pp. 309–318.

[9] JUNG, J., KRISHNAMURTHY, B., AND RABINOVICH, M. Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites. In *Proceedings of the International World Wide Web Conference* (May 2002), IEEE, pp. 252–262.

[10] RABINOVICH, M., AND AGGARWAL, A. RaDaR: A Scalable Architecture for a Global Web Hosting Service. *Computer Networks (Amsterdam, The Netherlands: 1999) 31*, 11–16 (1999), 1545–1561.

[11] RABINOVICH, M., XIAO, Z., AND AGGARWAL, A. Computing on the Edge: A Platform for Replicating Internet Applications. In *International Workshop on Web Caching and Content Distribution, Norwell, MA, USA* (2004), pp. 57–77.

[12] SLASHDOT. http://slashdot.org.

[13] SLASHDOT.ORG. Tanenbaum Rebuts Ken Brown, June 2004. http://linux.slashdot.org/article.pl?sid=04/06/08/1657256.

[14] SLASHDOT.ORG. Andy Tanenbaum Releases Minix 3, Oct. 2005. http://linux.slashdot.org/article.pl?sid=05/10/24/1049200.

[15] ZHAO, W., AND SCHULZRINNE, H. DotSlash: A Self-Configuring and Scalable Rescue System for Handling Web Hotspots Effectively. In *International Workshop on Web Caching and Content Distribution (WCW'04), Beijing, China* (October 2004), pp. 1–18.

[16] ZIPF, G. K. *Human Behaviour and the Principle of Least-Effort: An Introduction to Human Ecology*. Addison-Wesley, 1949.