WE NEED A MORE SOPHISTICATED WAY OF LOOKING THROUGH THE LARGE AMOUNT OF DATA COLLECTED AT EACH POINT TO DISCERN CORRELATIONS BETWEEN COMPONENTS.

This separation and replication of components allows Web services to scale in response to new resource demands. This is accomplished by introducing new servers hosting the particular, needed component. Despite this scalability, flash traffic patterns can drive a Web system's middleware component (or components) into overload. This leads to poor performance as the system is unable to keep up with the demands placed on it, and users see increased response times for their requests (see Figure 1). Experiments have indicated that users can tolerate roughly eight seconds of delay before they either retry their request or leave the site [2].

While the need for an admission control scheme is clear, formulating an effective system is daunting and error-prone. This is due to the large number of request types and middleware components. Different requests to a Web service stress different middleware components [1, 3]. It is advantageous to preferentially throttle those requests most correlated with the bottleneck. To do that, better visibility into the relationship between requests and their effects is necessary. Unfortunately, current system software and site-monitoring tools do not provide the operator with this visibility. For Web services to be more self-adaptive, they need to be more introspective, identifying correlated effects between internal components, so that the operator can act to shed load from overloaded components without penalizing all users of the Web site.

To design such self-adaptive Web services able to gracefully respond to overload, we propose four design mechanisms: simple statistical techniques for uncovering request effects in multi-tier systems; a black-box approach to middleware component monitoring; a visualization tool summarizing statistical findings to facilitate human decision making; and efficient techniques for operators to invoke admission control decisions based on those findings. We will also argue that including humans in the loop complements, rather than detracts from, self-adaptive design goals. We present ongoing work on a Web service based on the open source RUBiS auction site (see rubis.objectweb.org) that embodies these mechanisms. RUBiS is a Web-service benchmark designed to profile the performance of an auction site like eBay. Our approach leads to a Web service that is able to serve 70% more requests per second. Additionally, the maximum request latency seen by the user is reduced by 78%. These initial results show promise that middleware-based Web services can greatly benefit from more self-adaptive design.

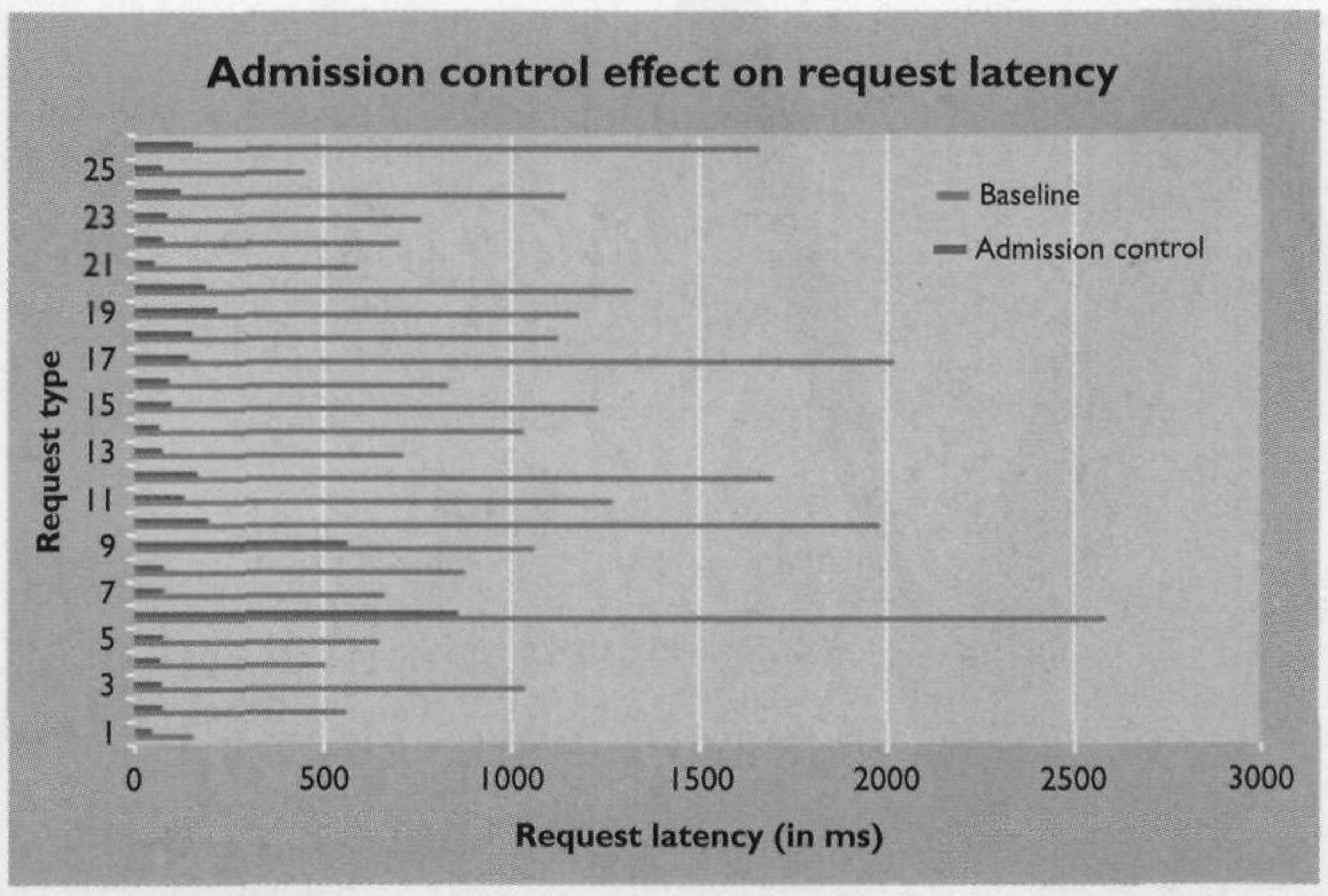

Figure 1. Selectively applying admission control to those requests correlated with the system bottleneck substantially reduces average request latency for the Web service.