Q) Different stages communicate with each other through event queues. How to pass variables (through reference, pointer, or copy)? If through reference or pointer, lock is needed to synchronize continuous stages and therefore performance will be reduced. Otherwise, copy will incur performance degradation when large amount of memory needs to be copied. Performance cost seems to be unavoidable between two adjacent stages.

A) Paper does not mention anything on variable passing among stages. Either it should design in a way that is not occurring (which is somewhat hard in a web server) so that you can pass the data through the queue. If it is not the case, yes I agree that a locking mechanism has to be there and it will be a costly.
Q) In the Haboob architecture, there are 10 stages. The author has not clearly stated the trade-off or consideration when designing stages. Because of the cost between stages, I doubt maybe 4 stages will be better than 10 stages. The 4 stages are socket listening/reading stage, http parse/fetch cache stage, socket write stage, file I/O stage. Unfortunately, there is not experiment to show how many stages are suitable for a web server?

A) Yes, the optimal number of stages is crucial part missing in the paper. However more the stages you have higher visibility you have on the stream and adaptive resource management would be more effective then again it will add additional latency of queueing delay and other costs involved.
Q) I don't think SEDA architecture show better scalability than other architecture. The experiment just shows that Haboob outperforms Flash and Apache in performance, not scalability.

A) In my point of view, it should be the opposite of that. SEDA do not out performs APACHE or FLASH but performs well under heavy workload. Thus it is more scalable than the later two options.
Q) Pipe line model and Master-Servant model are some popular models for multi-threaded programs. SEDA simply implementing a pipe line model thread architecture. I feel there is noting new in this paper other than a new implementation for high performance web server domain (probably). Because using pipeline model is a traditional way of doing thing in several high performance systems. What is you opinion regarding this?

A) What author tries to bring forward as the novelty in this, introduction of structured event queues (network of stages) along with dynamic resource controllers.
Q) How does the resource controller be sensitive to the resources of each stage? Is it reasonable for controller to monitor the runtime state of stages using local knowledge of a particular stage or global state? It seems that if the controller impedes too much of the execution of the stage, it may compromise the performance of it and also involve some security issues.

A) The controller does not intervene to the internal execution, what it monitors is the inputs in the input queue and the response (response time). There will be a cost in monitoring yet this is an important aspect in graceful degradation.
Q) If each stage randomises the order in which it processes I/O events delivered by the operating system, it seems that this system will not provide user-end programming with priority.

A) It will be the SEDA platform that is dealing with underline resources not the user programmer.
Q) The paper's idea seems to be practical and impose little overhead of operating system to support this mechanism. So has their work been adopted by the real Linux kernel? If not, Why?

A) Yes, there is one kernel called Vortex kernel implemented in 2003 and following are their conclusions.

1. The SEDA architecture also offers a solution to managing and implementing a complex event-driven design, and reduces the complexity of managing concurrency.
2. Our initial results indicate resource controllers can provide accurate information about system load.

There are couple of commercial and academic products based on this,

# LimeWire # TerraLycos # Rimfaxe Web Server
# Apache Excalibur Event Package
# SwiftMQ, a JMS Enterprise Messaging Server
# MULE Universal Message Objects,
# OceanStore, peer-to-peer filesystem
Q) The authors mention the idea of having controllers to manage how much of each type of resource is dedicated to each "stage". They mention the idea of controllers with global knowledge that can manage resources with an understanding of the system as a whole, but they don't actually discuss a concrete implementation of such a controller (as they do with the basic local-knowledge controllers). It's easy to imagine problems with local-only controllers: given a website with a variety of requests static and dynamic, say some requests will be satisfied with very little effort while others will require a lot of work. Using local knowledge only, more and more resources will likely be given to the stages that handle the expensive requests, which will bring the system down to a constant slow level of performance. It would be better to allocate some resources to the expensive requests while leaving a decent amount of resources for the cheap requests, thus satisfying the cheap requests very quickly. In a real-life system, where performance can't practically be predicted a priori, how would this be implemented?
A) If it going to work only using local knowledge, it will happen. That is where global knowledge is important. It should always consider the time to response compared with other stages. Which could be useful in deciding whether it should allocate more threads to this ... and there has to be a policy implemented on this governing the decision that should be take in such a scenario !!!
Q) The authors discuss "batching" of requests into the event handler. However, in the situations they discuss (a Web server and a packet forwarder), each request is independent. This suggests that all the event handler will do is iterate over the requests, handling each one in full before moving on to the next. What's the difference between handing the event handler a batch of events and having it iterate the batch, versus simply handing it one event at a time? It seems that the only difference between the two situations is the WHERE the loop is written, so why is there a difference in performance?

A) Batching requests means the number of requests handled by each thread in a given event. Higher the batching factor, more the chache locality but there will be high response time. Same time low batching factor would lead to low response time.
Q) In section 6 of this paper the authors state that event-driven programming is hard, since there are more tools and support for threaded programs, and that "many developers believe that event-driven programming is inherently more difficult". what happened between Ousterholt's talk ('95) and this paper ('01) to cause this?

A) Rather than a change in the landscape, this I will see more as author tries to set the platform for his project.
Q) Like capriccio, SEDA attempts to make decisions based on current resources usage. However, SEDA seems to use a simpler heuristic (SEDA scales the number of threads in a stage, and the number of events handled at the same time). Could SEDA do anything a bit smarter than this?

A) Yes, I think there can be many more complex decision making implemented but we have to think about the cost and time to take a decision as well.
Q) It seems that the adaptive adjusting of batching factor cannot make a stable output rate (as can be seen from Figure 9), I think the main reason would be resetting the parameter to the maximum value after a sudden drop in output rate, I'm wondering that would it be better that if we instead use a small decreasing of batching factor when sudden drop in output rate?
Q) In Figure 11, the SEDA based layer is compared against a thread based approach where threads are created when new connections come, I'm wondering that maybe this is not the best setting for thread based server architecture, and from the figure, what if we adopt the bounded thread pool approach, would it be better?

A) Then the question arise what is the optimal number for the pool size ...
Q) In case where the load is not that heavy, can SEDA have benefits against thread based approach?

A) This is mainly talking about graceful degradation in the event of a sudden hike ... so to which extent is it reasonable to measure that ...
Q) It seems like debugging a system like Haboob would be difficult, even though the authors claim that ordered events and thread isolation make it easier.

A) Debugging within a stage is again same as a conventional application but we can have a proxy stage in between two running stages and see what is going through.
Q) According to Matt Welsh's website, there are only a few systems where something SEDA like is being used. Does this mean there are non-hybrid systems that perform better now?

A) Quite interestingly, author later on admit few drawbacks of the system which he thinks lead to the fact that this is not widely used. However there is one occasion where another study has proven SEDA advantages over other systems.
Q) Isn't any event based system which does not somehow allow interruption or cancellation of long running events likely to fail in conditions when the length of each event in non-deterministic? In static pages this may not be the case, but in other cases the time for processing a read or write operation could vary vastly, making scheduling decisions difficult.

A) Yes, I think it's inherent issue we have in any event based system. In SEDA we can at least isolate such processes by having a finer level staging the web application itself.
Q) The paper does not talk about interactions between multiple threads working on the same queue. While it may be acceptable in the case of static pages to have no synchronisation between them, to scale the idea to dynamic pages or an OS wouldn't it be necessary to introduce some kind of synchronisation between them?

A) I think even handler process does the controlling the access to the queue, yet Yes it is necessary to introduce such a mechanism to manage shared objects among threads.
Q) Does the stages architecture of SEDA integrate FSM within it? Who keeps track of the continuation stage of a request?

A) In the architecture given, it has to be managed by the application.
Q) If a malicious user cheat to send a lot of requests, which will become events, how does SEDA deal with such situation? Is it necessary to add another controller to control this?

A) There is that vulnerability that a malicious user can exploit the fact that dynamic resource controllers will not check the authenticity of the requests.
Q) If the system is overloaded, is it better to serve a few clients quickly or is it better to treat them equally but slowly in order to show some sort of fairness?

A) Discussion - Totally depend on the nature of the site ...
Q) How to decide a discrete threshold value for resource controllers? I wonder if such case would happen: The threshold is always above the load, thus it would prevent the controller from adding new resources, even if the system is overloaded.

A) Value should be in a range such that even system reach that point it should be able to serve all the requests fairly.
Q) How much stuff from SEDA paper is actually been used in production now-a-days? If possible, discuss architecture of modern web servers like nginx and compare them to SEDA. (I bet that dynamic resource scheduling stuff is not being used, though it is a good idea!).

A) Client sends HTTP request -> Nginx chooses the appropriate handler based on the location config -> (if applicable) load-balancer picks a backend server -> Handler does its thing and passes each output buffer to the first filter -> First filter passes the output to the second filter -> second to third -> third to fourth -> etc. -> Final response sent to client
"In this paper we assume preemptive, OS-supported threads in an SMP environment, although this choice is not fundamental to the SEDA design."

Assume that there is no support for threads at kernel level, how are they going to handle this situations? processes with IPC :-) ? User-level threads won't help because their whole idea of using threads is to "provide nonblocking behavior on top of blocking OS calls by using a thread pool".
Q) Why not change the priority of each thread, on which each event monitor bases, in order to control dynamic resource allocation? I think it can reduce the overhead compared to controlling the number of threads on each event handler.

Q) SEDA uses the batch control as well as the thread pool control, based on the design that event handler and event monitor use separate thread, why not use a whole thread to simplify the design?