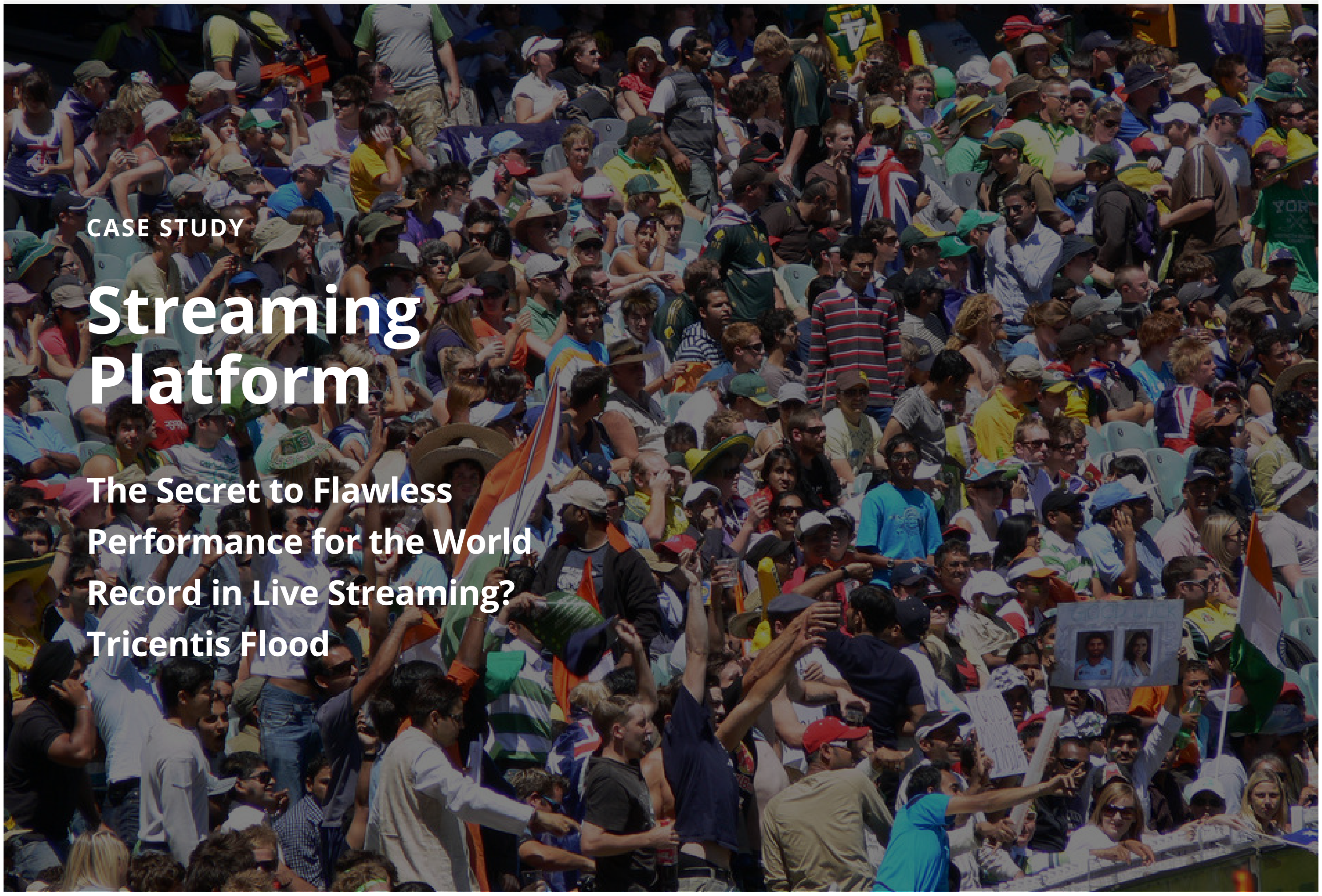

CASE STUDY

Streaming Platform

The Secret to Flawless Performance for the World Record in Live Streaming? Tricentis Flood

Case Study

The Secret to Flawless Performance for the World Record in Live

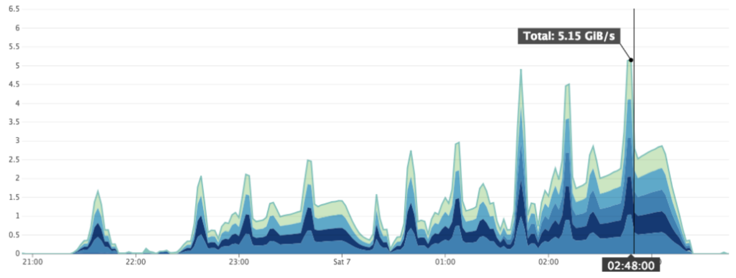
Streaming? Tricentis Flood

2018 was a banner year for India's largest streaming platform. The company was founded in 2015 as a mobile application for streaming the Cricket World Cup. Since then, it has grown into one of India's premier entertainment platforms, drawing a particularly large viewership for live-streamed Indian Premier League (IPL) Cricket matches.

The team celebrated a milestone during the live stream of the IPL 2018 season opener, hitting 4 million concurrent viewers – the most watched game to date on the platform. However, that milestone was blown away in April 2018 when a particularly exciting match reached 5.5 million concurrent viewers, making it the second most watched live event in worldwide streaming history.

By the end of the IPL 2018, the company had broken all records again with a peak concurrency of 10.3 million viewers – officially setting the world record in live streaming.

Successfully live streaming an event to 10.3 million concurrent viewers requires some serious load testing. The team put months of effort into testing, scaling, and planning for the big event, using Gatling and Tricentis Flood to make sure that their platform could perform under the demand.



In the words of their project lead:

"Our platform is the home of live cricket. Having done massive-scale cricketing events in the past, we were fortunate to have real game time traffic models to fall back on. Repeatedly, our older platform would come under duress as notifications were sent when something interesting happened. The ensuing spike in traffic would then overwhelm our back-end systems and occasionally disrupt video rendering. The legacy back-end architecture was unable to scale beyond a certain point, so this further limited us from bringing new customers onto the platform.

We learned to tackle the spikes the hard way. We learned to limit the use of auto-scaling to ensure that the right amount of servers always existed in a pool, but were always scaled up for the peak. We improved our utilization of our cloud provider by reviewing and tuning configurations, and we eliminated any clients that relied completely on the server systems to make decisions. We also made a dozen other changes, including early rejection of spurious traffic, tweaking bit-rate settings to adjust for latency, and implementing the golden rule of performance tuning: if it's not essential to do something real time, *don't* do it in real time.

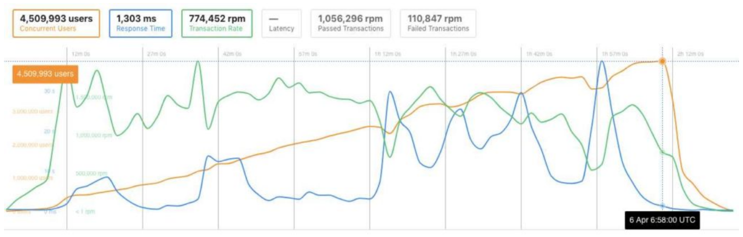
Once we figured out a way to test our systems' scaling using Tricentis Flood and Gatling, we held a series of "game-days": simulations of real IPL events, before they actually happened.

Our game-day simulations have three major components: a traffic model, the simulation script, and the load generator infrastructure. After determining our traffic model, we tuned our scripts with Gatling, mixing and matching user properties to generate a mix of user stories. We broke these scripts into modules, built our user journeys, and stitched everything together to form the simulation.

To run a million scale simulation, you need an army of load generators with aggregated reporting. This is not an easy task in itself. We looked at many available solutions like BlazeMeter, Gatling Frontline, etc. But we selected Tricentis Flood.

By the end of the IPL 2018, we were using a fleet of 600+ node grids across multiple regions, and had utilized more than 4000 computing hours. We ran 1000+ floods simulating 5 million concurrent users—with 2 million requests per minute and up to 5 Gbps of network traffic.

The game-day simulations helped our teams get used to how can event could play out, and the simulated spikes and ensuing mayhem helped us to tune our configurations as well as grow our operational muscle. As a result, our teams knew exactly what they had to do in a crisis, ahead of time, rather than try to figure it out "on-the-fly" when a true crisis arose."


4,509,993 users
Concurrent Users
1,303 ms
Response Time
774,452 rpm
Transaction Rate
—
Latency
1,056,296 rpm
Passed Transactions
110,847 rpm
Failed Transactions
12m 0s
27m 0s
42m 0s
57m 0s
1h 12m 0s
1h 27m 0s
1h 42m 0s
1h 57m 0s
2h 12m 0s
4,509,993 users
6 Apr 6:58:00 UTC