Issue: Batch processing system has a high failure rate and extremely long run times.

Description: Customer system is used multiple times per day to run batch processes of complex data calculations. Process is composed of web service calls, cross-domain data aggregation and calculations. Customer suspected a memory leak in batch processing was causing batch failures and extremely long run times. The recurring process failures were having a significant impact on business performance -- customer base would spend a significant portion of their day ushering batches through the system and waiting for completion.

Architecture: Thick client calling a web service for large batch processes. Traditional fixes have been to add additional memory to the application server and client hardware.

Batch system running average of 600 records
Average total computational time of 4h
End result, average total computational time of 1.5min

Resolution steps:

1. Started with memory profiler. Immediately saw that there were no memory leaks. Did however point to the most memory intensive class/method in the process.

2. Next identified the source of the client failures and determined they were also not caused by memory issues. We corrected the client interface with the web service to make it more resilient. As a result, the process would always run to completion with any individual process failures marked as such. Rather than failing the entire batch run, now the single record was marked as a system failure.

3. Developed a full set of integration test as a safety net to ensure any changes made to the web service were not corrupting the system results.

4. We introduced as series of timers into the code to see which methods in the process were taking the most time. This also gave us our baseline run numbers that we could use for A/B testing on system enhancements.

5. Using the timer results we focused on the longest running process first. We used a series of techniques to standardize the process domain, forward-stage data that was being frequently requested using caching, and eliminated redundancies. We systematically worked down the timer results from least performant to most in order to address each step in the batch process. The performance gain went from ~4 hours to ~17 mins

6. Further inspection of the client allowed us to parallelize the calls the web service. Doing so quickly reduced the remaining average run time from ~17 mins to ~1.5 mins.

Conclusion: Using an informed, surgical and systematic approach we were able to get the performance time down to ~1.5 min of processing time. We made these changes over the course of three weeks. The return on the customer's investment for these changes was achieved within a week. More importantly the customers have been freed to carry out other business tasks besides babysitting their systems.