performance requirements of ten users, while a large version of a system must provide them to meet the performance requirements of 100 users doing the same sort of work. Here, the dimension of scalability is the number of users, the context is the type of work they do, while the extent is ten users for the small system and 100 for the large one.

Impediments to load scalability include the occurrence of unproductive cycles (as in the case of busy waiting on locks to implement mutual exclusion rather than semaphores) and the inability to exploit parallel processing power because of serial or single-threaded processing of transactions that use disjoint data. Structural scalability can be constrained by the sizes of address spaces or of fixed-size sequence numbers. For example, the length of an array is constrained by the maximum bit length of its integer index. Questions about the impact of a design choice on load scalability can sometimes be answered by applying a simple analytic model over a wide variety of parameters. We have done this to justify the use of semaphores rather than locking instructions 1 implement mutual exclusion [3].

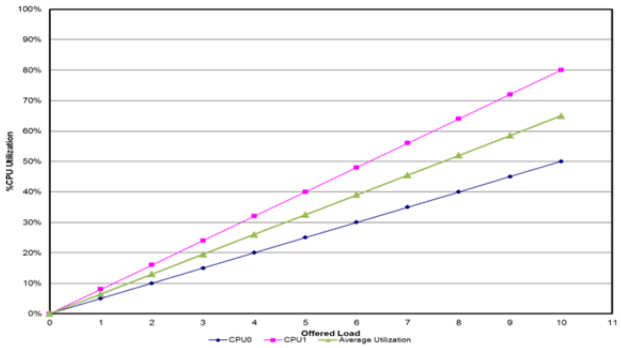


**Figure 2. Unbalanced CPU loads due to serial execution via single thread.**

Measurements taken during performance tests can also reveal limitations on load scalability [4]. Figure 2 illustrates contrived data representing measurements of a two-processor UNIX™-based system. The data are contrived because measurements of the actual system were not available for publication. The straight line between the upper and lower lines shows the average CPU utilization overall. Observations of the actual system were taken with *mpstat.* An examination of *ps –eLF* output taken at regular intervals revealed that activity was concentrated in two processes for which changes in the cumulative processing times corresponded to the two processor utilizations. Since cache affinity was turned on, we inferred that each process was bound to one processor, yielding test results like those shown in Figure 2. The developers explained that the more CPU-intensive process was processing disjoint sets of data sequentially. An opportunity for parallel execution had been overlooked. The maximum offered load of the system was constrained by the larger of the two CPU utilizations. Thus, the processor imbalance limited the extent of load scalability to 10 work units in unit time, while architecting the system to allow parallel execution on disjoint data sets could have increased the extent of load scalability to 12.3 units of work in unit time, in the absence of any other bottlenecks.

# 4. CONCLUSION

The foregoing discussion and examples illustrate how performance engineering methods can be applied to mitigate performance risk throughout the software life cycle. Clearly specified performance requirements, performance-oriented architecture reviews, and performance tests planned in the light of requirements all contribute to the mitigation of performance risk and the delivery of a product that meets performance expectations.

# 5. ACKNOWLEDGMENTS

The figures in this paper originally appeared in [4]. They are reproduced with the permission of the publisher. The author has benefited from frequent discussions with Alberto Avritzer and Bob Schwanke.

# 6. REFERENCES

[1] Avritzer, A., and A. B. Bondi. 2012. Resilience assessment based on performance testing. In *Resilience Assessment and Evaluation of Computing Systems*, edited by K. Wolter, A. Avritzer, M .Vieira, and A. van Morsel. Springer.

[2] Avritzer, A., and E. Weyuker. 1995. The automatic generation of load test suites and the assessment of the resulting software. *IEEE Trans. Softw. Eng.* 21(9), 705–716.

[3] Bondi, A. B. 2000. Characteristics of scalability and their impact on performance. In *Proc. WOSP2000,* Ottawa.

[4] Bondi, A. B. 2014. *Foundations of Software and System Performance Engineering.* Addison-Wesley, Upper Saddle River, NJ. ISBN-13: 978-0321-83382-2.

[5] Browne, J.C. 1981. Designing systems for performance. Keynote address, ACM SIGMETRICS Conference, Las Vegas, Nevada, 1981. In *Performance Evaluation Review* 10 (1), 1, 1981.

[6] Denning, P. J., and J. P. Buzen. 1978. The operational analysis of queueing network models. *ACM Computing Surveys* 10(3), 225–261.

[7] Eilperin, J.2013. CGI warned of HealthCare.gov problems a month before launch, documents show. *Washington Post,* October 29, 2013.

[8] IEEE Std 830-1998, *IEEE Recommended Practice for Software Requirements Specifications -Description.*

[9] Masticola, S., A. B. Bondi, and M. Hettich. 2005. Model-based scalability estimation in inception-phase software architecture. In *Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science* 3713, 355–366.

[10] Schwaninger, C., Wuchner, E., and Kircher, M. 2004. Encapsulating crosscutting concerns in system software. Proc. Third AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software.

[11] Smith, C.U., and Williams, L.G. 2002. *Performance Solutions.* Addison Wesley, Boston, MA.