HEWLETT PACKARD
COMPUTER SYSTEMS CENTER
HEWLETT PACKARD LABORATORIES

15 April 1988

Position statement for ACM SIGOPS European Workshop 1988

John Wilkes

It's my belief that the issue of "autonomy vs interdependence" is at least as much a sociological, political, and budgetary question as it is a technical one. Notwithstanding this, the problem is still an interesting technical one from two points of view: what functions to provide to meet best the range of sociological, political, ... pressures, and what are the technical issues in providing these services.

In order to discuss the choices better, we probably need a classification for "autonomous systems" of some sort. Here's one such:

1. Independent (fully autonomous) systems. These crop in a large number of situations, none of which are particularly of interest here.

2. Occasionally dependent systems, that once in a while (but it doesn't matter very much when in real time) will connect to, and take advantage of, some external services. Examples here include portable laptop machines, where it is acceptable to operate in an impoverished environment as an alternative to not doing work at all. These systems aren't particularly interesting here either: resource access is typically handled on a fairly leisurely basis, and remote utilization of the laptop is not possible (or helpful).

3. Dependent systems, which import services (function, data, ...) in real time, but where some work can continue even if the remote services fail, but the working environment will be noticeably impaired if this continues for any length of time. It seems that most "workstation users" fall into this category: they need remote dumping, printing, file access, name lookup, etc services, but can survive for short periods of time (typically from a couple of minutes to a few hours) with partial or complete disruption of access to remote services.

4. Partially interdependent systems, where some resources are exported, in addition to the use of imported ones. Obviously, the servers used by the dependent systems belong here; but there is a growing trend towards using other machines (particularly so-called "idle resources") to offload work from a busy node, or shorten the response-time properties of some task.

5. Fully-interdependent systems, with no local autonomy. These typically crop up in machines constructed to provide attributes of higher performance than can be obtained from a single processing node, when incrementally scalable systems are important, and when high availability ("fault tolerance") is required. Here there is no issue about ownership of individual components, and thus no real question about degrees of autonomy, other than as needed to provide fault isolation and containment.

It's my belief that the issues of "autonomy vs interdependence" only appear in levels 3 and 4 of the classification above, and to a large degree are research problems because of the emotional reactions that class 3/4 usage brings to "owners" of the resource providers. Given a fixed amount of money (or equivalent), and a set of users, different answers will arise depending on their approach to treating a set of machines as a common pool or as a set of privately owned systems.

It's important to distinguish "ownership" from placement. For example, my group disperses its more powerful machines around our offices because it simplifies the electrical power distribution; nobody cares (or knows, frequently) what one of the machines in their office is being used for, and no "ownership" is felt for them, so the question of autonomy doesn't arrive.

Our experience has been that a good way to approach autonomy issues is to first remove the barriers to fully interdependent operation, and then selectively re-impose autonomy options, made available to "owners" of resources.

Interdependence design issues at HP Labs

The computing environment for the group in which I work consists of a large number of workstations and larger computing engines (roughly 30 machines between the 10 people in the group), of a wide range of different system types and capabilities. We are part of a much larger internetwork: some several hundred machines on the local site, and a few thousand throughout the company. We take daily advantage of services provided throughout HP (for example, remote software installation to and from HP Labs, Bristol).

Some machines act as servers to the others—these fall into the "must be up" category; others are "home" machines for individual users, repositories for mail, private work, environment customization files, and the like. Others run discless as part of a cluster of machines, which all share a single seamless file system: these are used variously for testing purposes and as low-cost X servers. Still others run (and crash) experimental operating systems or application software, and are treated as a pool of machines made available to the software developers in either "reserved" or "shared" modes.

In this environment, the tension between autonomy and interdependence is a factor in our daily decision making. Here are some of the observations (and principles) we have found useful in choosing practical solutions to autonomy questions:

1. In an environment where at least some of the systems are "dependent" in the sense defined above, or more interdependent, there will almost always be a set of logically centralized services that fall into a "must be maintained" category.

    - The network itself: its internal cabling, repeaters, bridges and gateways. (It happens that some rather important services live on the other side of a gateway. The fact that we accept this is testimony to the effectiveness of the support environment in keeping the world in good shape.)
    - Cluster-root (disc) servers, which need to be up in order to run the discless workstations
    - A (powerful) timesharing machine used as a "home" machine for a number of users in the group, and as a general computation engine.

    Failures of any of these machines/subsystems are noticed in minutes. And fixed on similar timescales, too. In practice, this isn't burdensome – our crucial services simply don't break.

2. Several services can be replicated fairly simply, so MTTRs of a day or so are acceptable. Examples include name servers; network printers; NFS servers for rarely-used files; central "batch" software servers (more on this below); and individual discs, workstations or X window servers that can simply be swapped out for a working machine. Again, we observe small hardware mortality rates for such machines; the software failures can typically be handled trivially (e.g. restart), or ignored by accessing another server, typically automatically.

3. A powerful principle that we've applied in several instances is that the "recipient should be in control". This is most visible in our software update mechanism/policy: every night, our machines go out to a set of places and get the most recent versions of various pieces of software, configuration files, etc. (You can think of this as a reverse-rdist: only the minimum number of updates actually occur.) Since this action is performed by the clients of the distribution service, they can choose to pick up the latest updates or not, as they wish. At its best, it allows complete system upgrades (of scale comparable to moving from 4.1BSD to 4.3BSD) to be completed across the network in less time than it would take to mount and unload the tapes, and keeping all the local customizations in place.

   We take this principle still further: the installation process for the system by which this is all performed is itself installed the same way (once the installation program has been acquired). That software can itself be locally customized to accept or reject particular portions of the updates, leaving a great deal of control in the hands of the local machine owner. (In our case, "shared" machines get everything, but a few individuals choose that "their" workstations suppress some of the services made available this way.)

   This class of service is in the "batch mode" category: it doesn't matter if an update fails one night. At worst it can be run manually the next morning, but in practice it gets done automatically the next evening, and nobody minds.

4. We have found that a hierarchical delegation of autonomy is a very useful concept. In our current (experimental) implementation, we have three levels:

   - *standard*, which we expect (and make) completely non-controversial, so that it can be disseminated to a very wide user community (typically 200+) without compunction or nervousness on their part;
   - *group*, which is an agreed-upon set of additional functions or choices appropriate to a particular work group (in our case, this coincides with the union of two organizational entities);
   - *local*, on a per-machine basis.

   All the software that is distributed uses the union of the information in the *standard*, *group*, and *local* classifications in some application-specific way. Sometimes this is simple: the nightly installation process constructs an `/etc/printcap` file from the concatenation of the three components. Sometimes more careful processing is required, as in the pattern matching rules used to select access rules for remote resource usage on the local machine, or the selection and filtering process for the files to be backed up across the network. Application-specific defaults handle cases where one or more of the three components is missing or has been deleted.

   Obviously, this could be trivially extended to an arbitrary depth.

5. Local hosts can (and do) choose to protect themselves against mistakes that get propagated via the installation process, and refuse to accept the update. For example, our "group" server machine needs to stay up with very high probability, so takes care to reject suspicious-looking updates to important system components. (For example, if certain files shrink by more than a certain fraction of their previous length, it requests human assistance instead of accepting them.)

6. Social pressures are surprisingly effective mechanisms to achieve a comfortable steady state (once somebody abuses something in a large user community, they *never* do it

again!). Honest errors and teething troubles are the commonest (only?) causes of problems for us. This typically happens when new people are first exposed to the (rather complex because rather functional) environment in which we work. This means that autonomy as a protection mechanism against accidental damage is useful, but it seems to be unnecessary (at least with our user community) to adopt a more paranoid stance.

Conclusion

In conclusion, we feel that the degree of autonomy or interdependence considered desirable is largely a social, political and budgetary question. In our experience, desirable design points include a range of options, tailored to individual or group needs, and provision of a range of alternatives can be accommodated quite easily with uniform adoption of a few key principles. This workshop looks to be a good opportunity to share a few of these in greater detail, and to learn more about design issues for other environments.

Hewlett-Packard Laboratories
1501 Page Mill Road
P.O. Box 10490
Palo Alto, Ca 94303-0971
Tel: +1 415 857 3568