A Software Rejuvenation Framework for Distributed Computing

This framework supports graceful degradation of services at best possible performance levels.

NASA's Jet Propulsion Laboratory, Pasadena, California

A performability-oriented conceptual framework for software rejuvenation has been constructed as a means of increasing levels of reliability and performance in distributed stateful computing. As used here, "performability-oriented" signifies that the construction of the framework is guided by the concept of analyzing the ability of a given computing system to deliver services with gracefully degradable performance. The framework is especially intended to support applications that involve stateful replicas of server computers.

Software rejuvenation has been recognized as a simple yet effective means of preventing accumulation of software errors that, if allowed to accumulate, could degrade the capacity or cause failure of a computer system. When a software system is voluntarily rebooted, with high probability, errors accumulated during previous execution are eliminated and the system regains its full capacity. Although software rejuvenation has been investigated extensively, it has not, until now, been considered for stateful applications that involve server replicas. The problem of software rejuvenation in such applications is complicated by the following considerations: When software rejuvenation temporarily stops a long-running replica server, R, the post-rejuvenation performance of R may be reduced because the stoppage may cause the state of R to become inconsistent with the nominal state of other replicas. In that case, R would be unable to provide services at its full capacity until consistency with the states of the other replicas was restored.

The present performability-oriented framework is based on three building blocks: a rejuvenation algorithm, a set of performability metrics, and a performability model. The performability metrics and model both take account of the reduced nature of post-rejuvenation performance pending restoration of consistency. The performability model also takes account of the possibility that post-rejuvenation consistency-restoration processes could be vulnerable to failures because of the potential performance stress caused by service requests accumulated during rejuvenation.

The basic version of the rejuvenation algorithm uses pattern-matching mechanisms to detect pre-failure conditions. To compensate for the inability of pattern-matching mechanisms to detect pre-failure-condition patterns other than those known *a priori*, an enhanced version of the algorithm accommodates a random timer and provides for synergistic coordination of both detection-triggered and timer-triggered rejuvenation. It has been demonstrated, via model-based evaluation, that this performability-oriented framework enables error-accumulation-prone distributed applications to continuously deliver gracefully degradable services at the best possible performance levels, even in environments in which the affected systems are highly vulnerable to failures. It has also been shown that software rejuvenation can be realized as an integral part of the infrastructures in stateful distributed computing applications that guarantee eventual consistency of the states of server replicas.

This work was done by Savio Chau of Caltech for NASA's Jet Propulsion Laboratory.

The software used in this innovation is available for commercial licensing. Please contact Karina Edmonds of the California Institute of Technology at (626) 395-2322. Refer to NPO-42352.

Kurtosis Approach to Solution of a Nonlinear ICA Problem

A gradient-descent algorithm minimizes the kurtosis of an output vector.

NASA's Jet Propulsion Laboratory, Pasadena, California

An algorithm for solving a particular nonlinear independent-component-analysis (ICA) problem, that differs from prior algorithms for solving the same problem, has been devised. The problem in question — of a type known in the art as a post nonlinear mixing problem — is a useful approximation of the problem posed by the mixing and subsequent nonlinear distortion of sensory signals that occur in diverse scientific and engineering instrumentation systems.

Prerequisite for describing this particular post nonlinear ICA problem is a description of the post nonlinear mixing and unmixing models depicted schematically in the figure. The mixing model consists of a linear mixing part followed by a memoryless invertible nonlinear transfer part. The unmixing model consists of a nonlinear inverse transfer part followed by a linear unmixing part. The source signals are recovered if each operation in the unmixing sequence is the inverse of the corresponding operation in the mixing sequence.

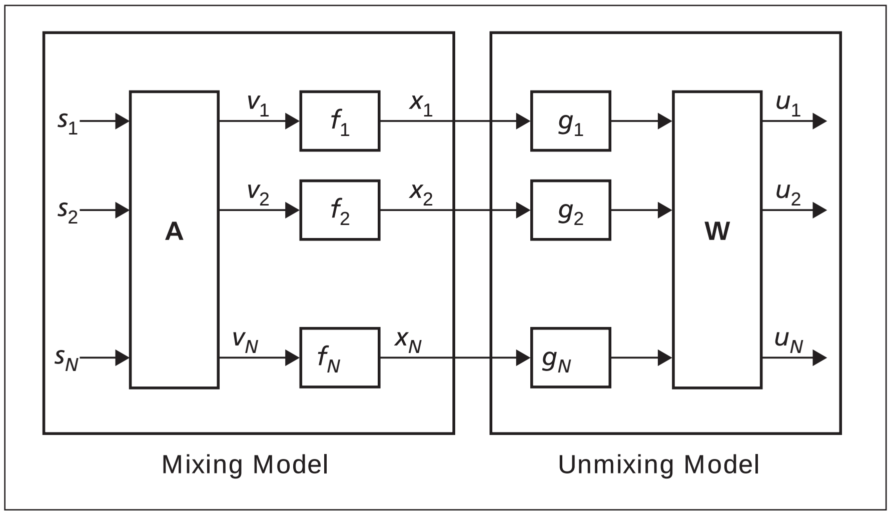
More specifically, in the models,

$$\mathbf{s}(n) = [s_1(n), s_2(n), \ldots s_N(n)]^T$$

is an $N \times 1$ column vector representing N independent source signals at time n that one seeks to estimate. This vector is multiplied by $\mathbf{A}$, an initially unknown $N \times N$ matrix that represents the linear mixing of the source signals. The N signals resulting from the mixing are represented by $N \times 1$ column vector

$$\mathbf{v}(n) = [v_1(n), v_2(n), \ldots v_N(n)]^T.$$

Each of these signals is then subjected to nonlinear distortion represented by a function that is initially unknown and could differ from the functions that represent the distortions of the other sig-



Mixing and Distortion Operations and their inverses are represented in these block-diagram representations of mixing and unmixing models.

nals. For the *i*th mixture signal, the distorted signal is given by $x_i = f_i(v_i)$, where f_i is one of the initially unknown nonlinear functions. Thus, the vector

$$\mathbf{x}(n) = [x_1(n),x_2(n),...x_N(n)]^T$$

represents instrumentation signals presented for analysis. The distortion in signal x_i is removed by means of a corresponding initially unknown inverse nonlinear function g_i. Finally, the signals are unmixed by means of initially unknown matrix **W** to obtain output vector

$$\mathbf{u}(n) = [u_1(n),u_2(n),...u_N(n)]^T.$$

In the ideal case, **W** would be the inverse of **A** and the output vector **u** would equal the vector, **s**, of source signals.

The particular nonlinear ICA problem is to calculate the nonlinear inverse functions g_i and matrix **W** such that **u** calculated by use of them is a close approximation of **s**. For the purpose of the present algorithm for solving this problem, it is assumed that the inverse nonlinear functions g_i are smooth and can be approximated by polynomials. The algorithm finds the components of the unmixing matrix **W** and the coefficients of the polynomial approximations of g_i by a gradient-descent method. This algorithm utilizes the kurtosis of the components of the output vector **u** as an objective function (in effect, an error measure) that it seeks to minimize. In using the kurtosis, this algorithm stands in contrast to prior algorithms that utilize other objective functions, including statistical functions other than the kurtosis.

This work was done by Vu Duong and Allen Stubberud of Caltech for NASA's Jet Propulsion Laboratory.

In accordance with Public Law 96-517, the contractor has elected to retain title to this invention. Inquiries concerning rights for its commercial use should be addressed to:

Innovative Technology Assets Management
JPL
Mail Stop 202-233
4800 Oak Grove Drive
Pasadena, CA 91109-8099
(818) 354-2240
E-mail: iaoffice@jpl.nasa.gov

Refer to NPO-43088, volume and number of this NASA Tech Briefs *issue, and the page number.*

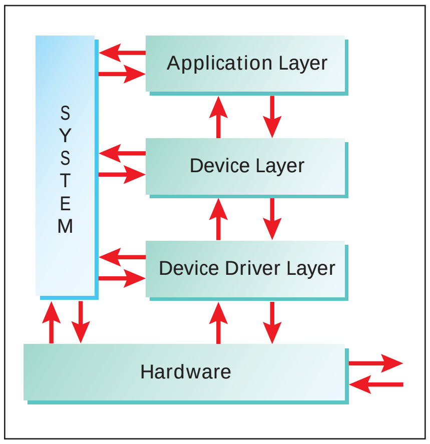
Robust Software Architecture for Robots

Generalized software can be readily tailored for specific applications.

NASA's Jet Propulsion Laboratory, Pasadena, California

"Robust Real-Time Reconfigurable Robotics Software Architecture" ("R4SA") is the name of both a software architecture and software that embodies the architecture. The architecture was conceived in the spirit of current practice in designing modular, hard, real-time aerospace systems. The architecture facilitates the integration of new sensory, motor, and control software modules into the software of a given robotic system. R4SA was developed for initial application aboard exploratory mobile robots on Mars, but is adaptable to terrestrial robotic systems, real-time embedded computing systems in general, and robotic toys.

The R4SA software, written in clean ANSI C, establishes an onboard, real-time computing environment. The R4SA architecture features three layers: The lowest is the device-driver layer, the highest is the application layer, and the device layer lies at the middle (see figure).



The **R4SA Architecture** features three levels corresponding to different levels of abstraction.

The device-driver layer handles all hardware dependencies. It completely hides the details of how a device works. Activities directed by users are performed by means of well-defined interfaces. Each type of device driver is equipped with its own well-defined interface. For example, the device-driver interface for an analog-to-digital converter differs from that for a digital-to-analog converter.

The device layer provides the means for abstracting the high-level software in the application layer from the hardware dependencies. The device layer provides all motion-control computations, including those for general proportional + integral + derivative controllers, profilers, controllers for such mechanical components as wheels and arms, coordinate-system transformations for odometry and inertial navigation, vision processing, instrument interfaces, communication among multiple robots, and kinematics for a multiple-wheel or multiple-leg robot.