Nuevas tendencias en tolerancia a fallos para aplicaciones paralelas

María J. Martín

Grupo de Arquitectura de Computadores - CITIC
Universidade da Coruña
mariam@udc.es

COMPUTER ARCHITECTURE GROUP
UNIVERSITY OF A CORUÑA
GAC.UDC.ES

citic

UNIVERSIDADE DA CORUÑA

1 **Introducción**
- Contexto
- Fallos en sistemas actuales
- Objetivos

2 Soluciones *Stop-and-restart*

3 Aplicaciones MPI Resilientes

4 Conclusiones

Supercomputadores actuales

- Clusters de procesadores multinúcleo/GPUs
- Alto poder de cómputo $\sim 10^{15}$ FLOPS
- Miles de nodos y miles/millones de núcleos

Summit: Nº 1 lista top 500

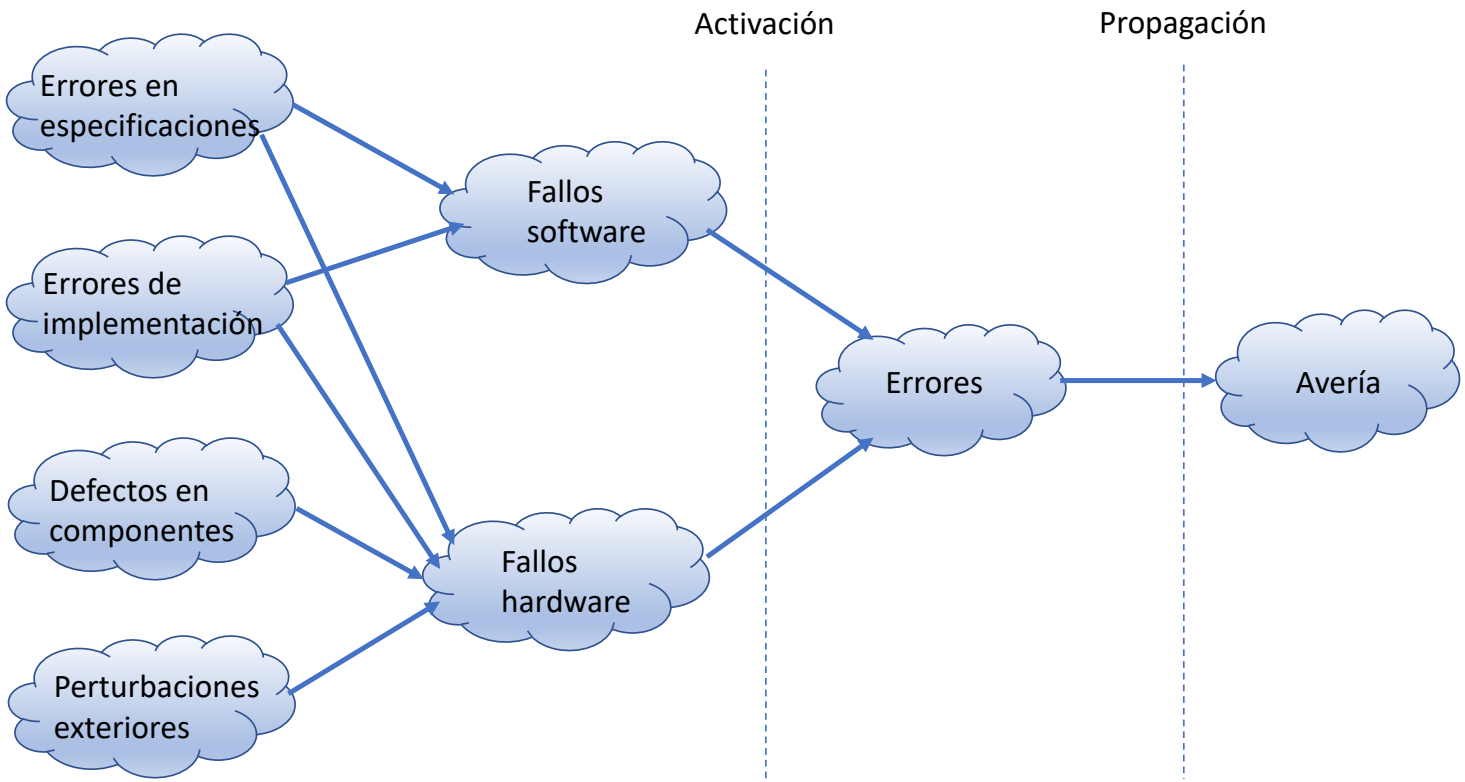
- 36.000 procesadores
- Más de 2 millones de núcleos
- 200 PFLOPS

Era exaescala

- Demandas computacionales continúan creciendo
  - Nuevos problemas
  - Big Data
- Próximo reto: computadores exaescala (10^{18} FLOPS)
  - Principal limitación: consumo energético
- Gran oportunidad para aplicaciones HPC, pero también un riesgo
- Sistemas más grandes y más complejos $\Rightarrow \downarrow\downarrow$ MTTF

MTTF (Mean Time To Failure): Tiempo medio hasta fallo

Fallos en sistemas actuales



- Atendiendo duración: transitorios, intermitentes o permanentes

Causas de fallos

- ↓↓ Tamaño transistores ⇒↑↑ Fallos hardware
  - Radiación cósmica
  - Variaciones en la manufactura
  - Envejecimiento más rápido
- ↑↑ Complejidad programación ⇒↑↑ Fallos software
- ↑↑ Nº nodos ⇒↓↓ MTTF
  - Nodo de computación: 1 fallo cada siglo
  - Sistema con 100k nodos: 1 fallo cada 9 horas

Blue waters

- Di Martino, C., Kalbarczyk, Z., and Iyer, R. (2016). Measuring the resiliency of extreme-scale computing environments. In Principles of Performance and Reliability Modeling and Evaluation (pp. 609-655). Springer, Cham.
  - 25k nodos, 380k núcleos, 11.6 PFLOPS
  - Aplicaciones fallidas (261 días): 9% total horas consumidas
  - Coste en electricidad: 1/2 millón dólares

- Futuros sistemas exaescala: tasa de fallos mayores
- Aplicaciones largas necesitan tolerancia a fallos
- **Tolerancia a fallos:** capacidad de continuar funcionando después de un fallo

MPI: estándar de facto aplicaciones HPC en sistemas distribuidos

Objetivo

- Aplicaciones MPI tolerantes a fallos
- Consideraremos:
  - Fallos hardware o software que interrumpen la ejecución de la aplicación

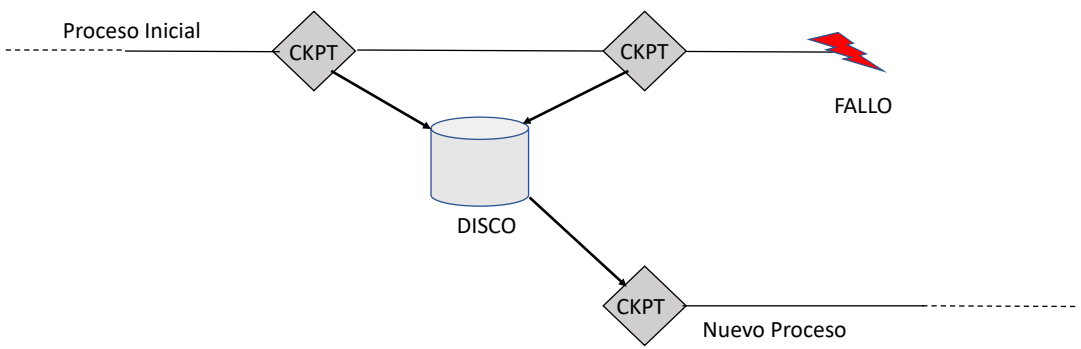
1. Introducción
2. **Soluciones** *Stop-and-restart*
   - Checkpointing
   - Herramienta de checkpointing CPPC
3. Aplicaciones MPI Resilientes
4. Conclusiones

Stop-and-restart

- MPI carece de soporte para tolerancia a fallos
- Solución tradicional: Stop-and-restart basado en checkpointing

Checkpointing

- Guarda de forma periódica el estado de la aplicación
- En caso de fallo, la aplicación continúa la ejecución desde el último estado guardado



Características del checkpointing

- Granularidad, Transparencia, Coordinación

Granularidad

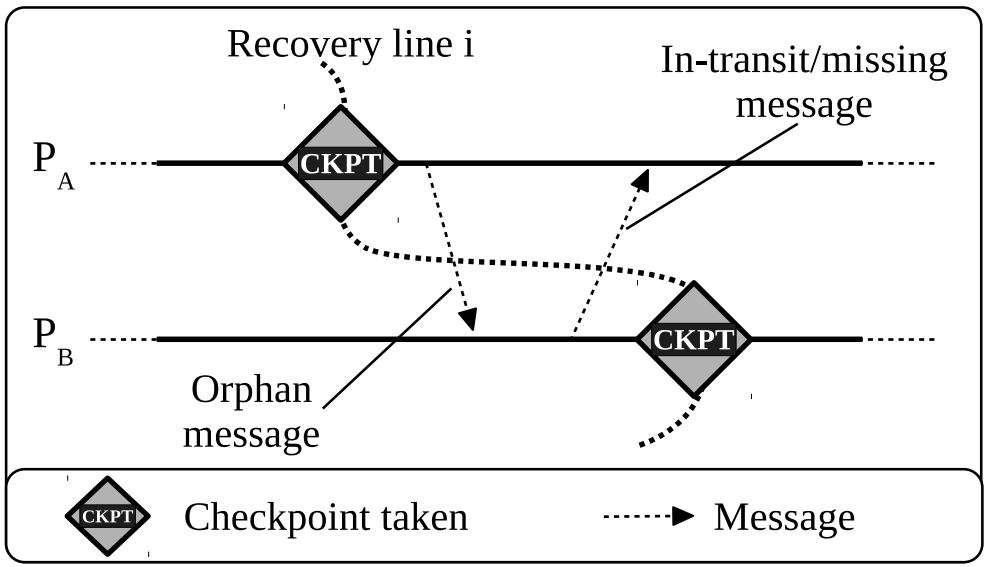
- A nivel de sistema:
  - Se almacena el estado completo
  - Transparente, sobrecarga alta, no portable
- A nivel de aplicación:
  - Se pueden almacenar solo datos necesarios
  - Mayor rendimiento, se necesita analizar el código

Transparencia

- Forma en la que se percibe la técnica a nivel de usuario

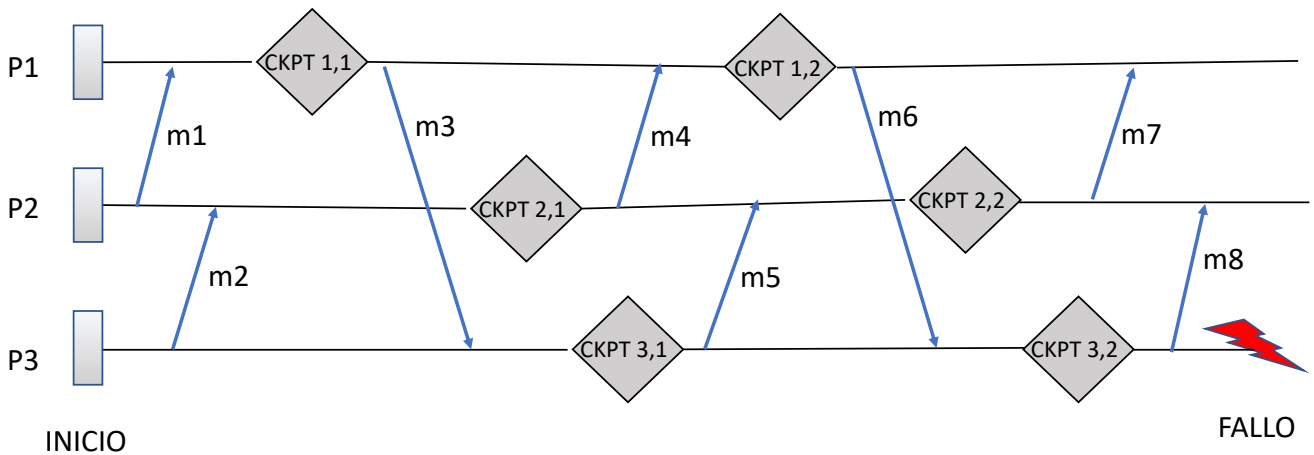
Coordinación

- Las aplicaciones paralelas necesitan coordinarse para evitar inconsistencias debido a las comunicaciones
  - Mensajes en tránsito: enviados pero que no han sido recibidos
  - Mensajes inconsistentes: recibidos pero que no han sido enviados



Protocolos de coordinación

- No-coordinados
  - Baja sobrecarga en una ejecución sin fallos
  - Susceptibles de *efecto dominó* en reinicio
  - Se pueden combinar con logging de mensajes

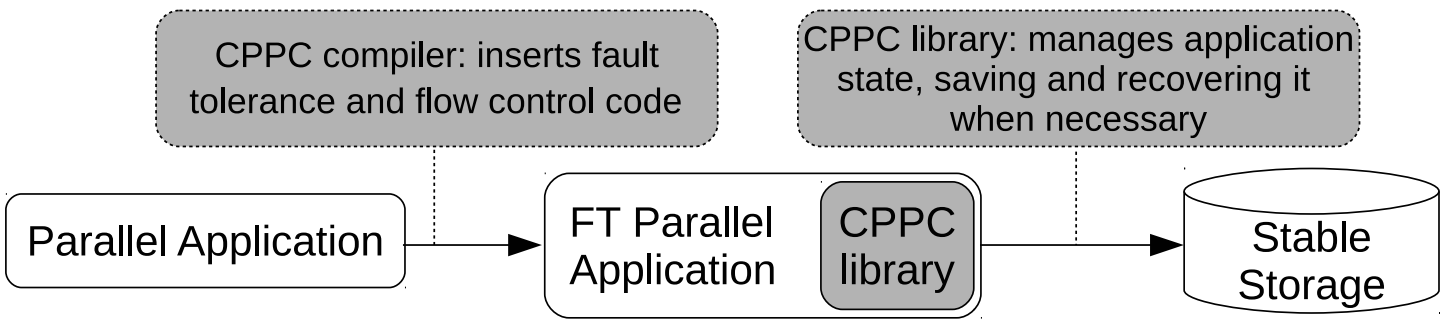


Protocolos de coordinación

- Coordinados
  - Los procesos se sincronizan para volcar un estado global
  - **Ventajas:** Se simplifica el reinicio
  - **Inconvenientes:** Mayor sobrecarga en ejecución sin fallos
  - Dos implementaciones posibles:
    - Bloqueante: Requiere la interrupción de la computación
    - No bloqueante: No requiere la interrupción de la computación

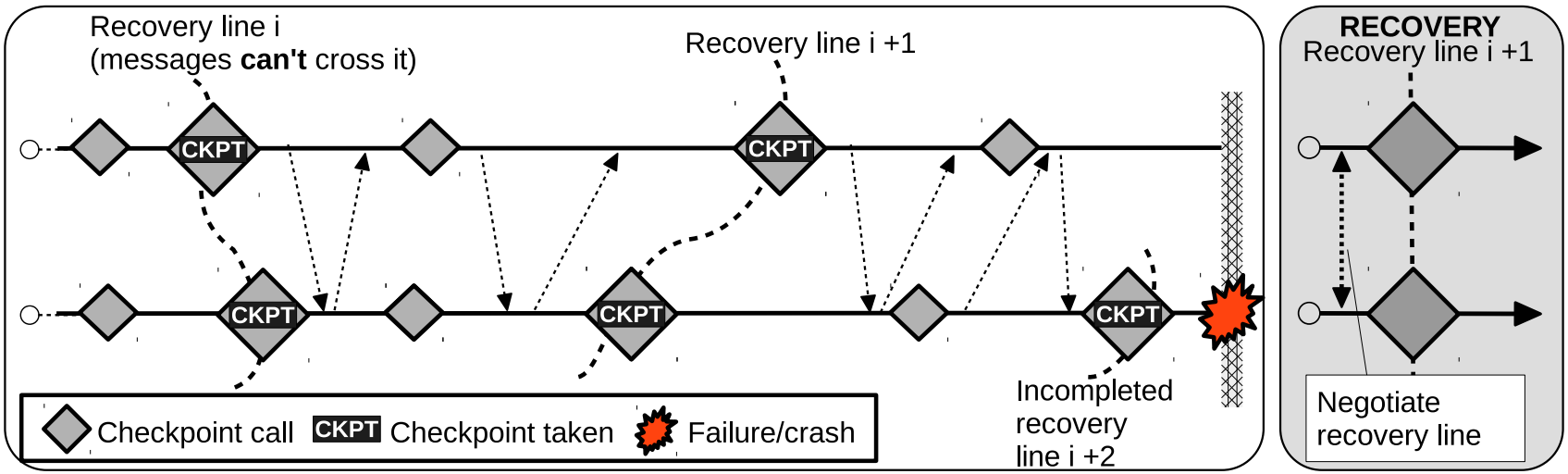
CPPC — Librería + Compilador

- El compilador instrumenta el código con llamadas a la librería
  - Registra las variables relevantes
  - Inserta llamadas de checkpoint
    - En puntos seguros para asegurar la consistencia
    - En los lazos más costoso (cada N llamadas) para tener una frecuencia adecuada
- Volcado multi-hilo: solapa E/S y computación



Protocolo de coordinación espacial

- Procesos hacen checkpoint de forma independiente → NO sincronizaciones en tiempo de ejecución
- Llamadas de checkpoint en puntos seguros → comunicaciones no pueden cruzar la línea de recuperación



Puntos fuertes CPPC

- Protocolo coordinación espacial
- Tamaño ficheros checkpoint
  - Otras alternativas para reducir tamaños: checkpointing incremental, compresión, ...

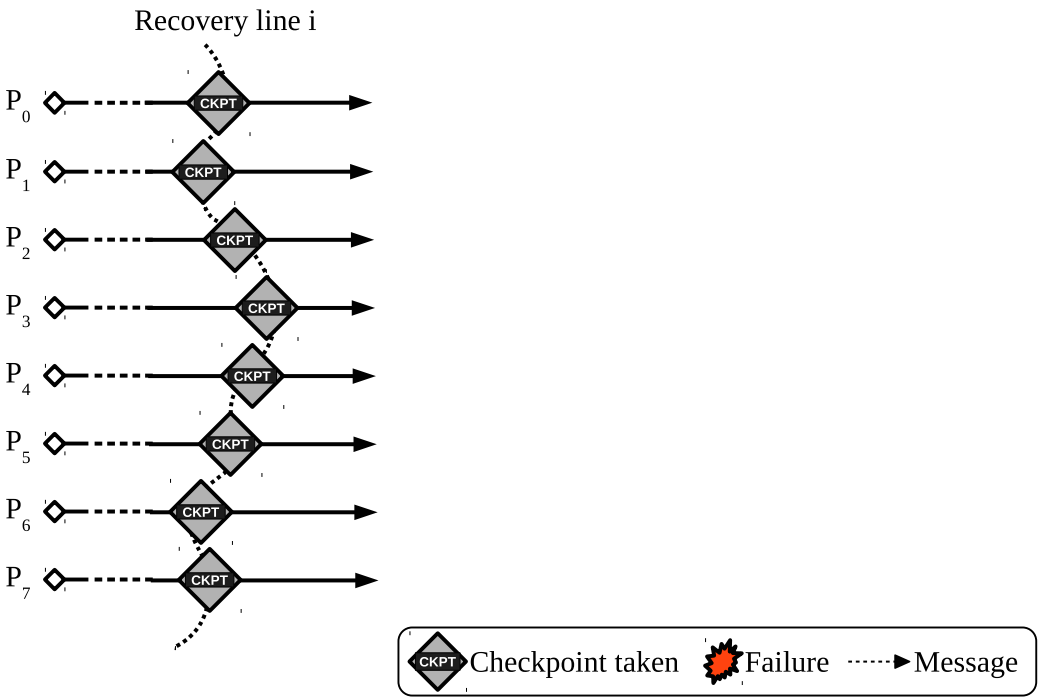
1 Introducción

2 Soluciones *Stop-and-restart*

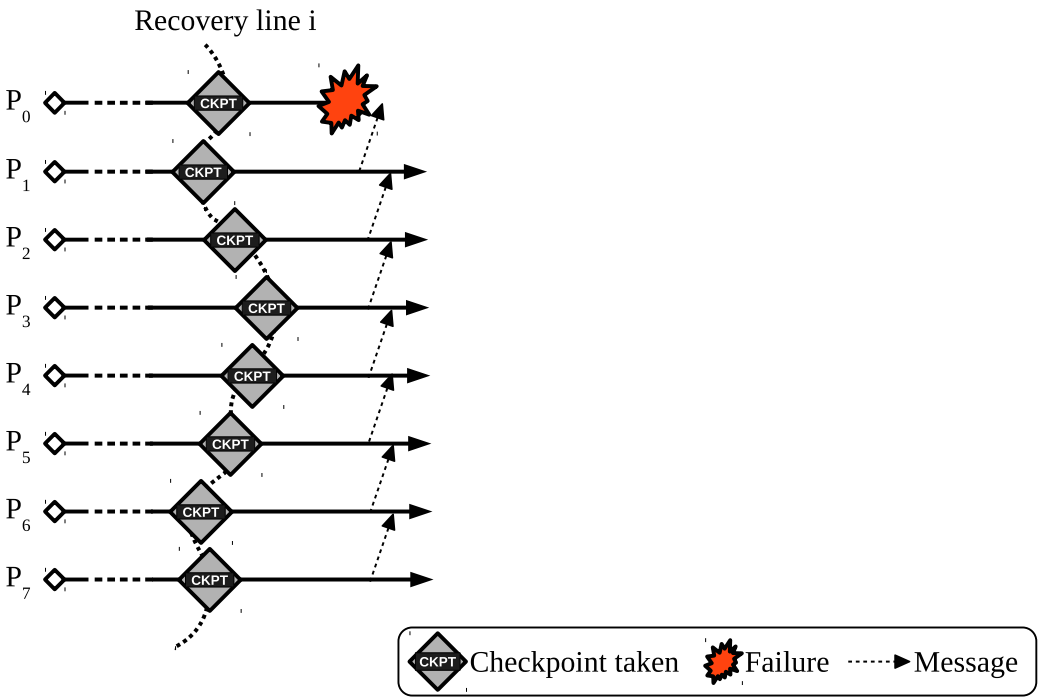
3 **Aplicaciones MPI Resilientes**
- Introducción
- User-level Failure Mitigation (ULFM)
- Aplicaciones MPI resilientes usando ULFM
- Nuestra propuesta
- Evaluación experimental

4 Conclusiones

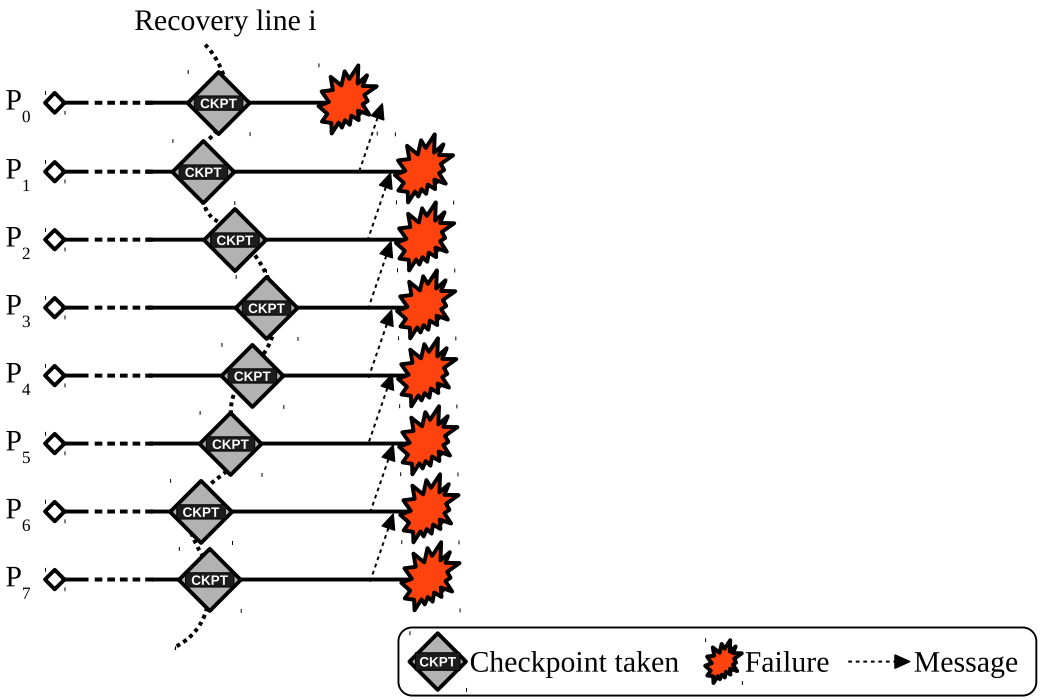
Soluciones stop-and-restart



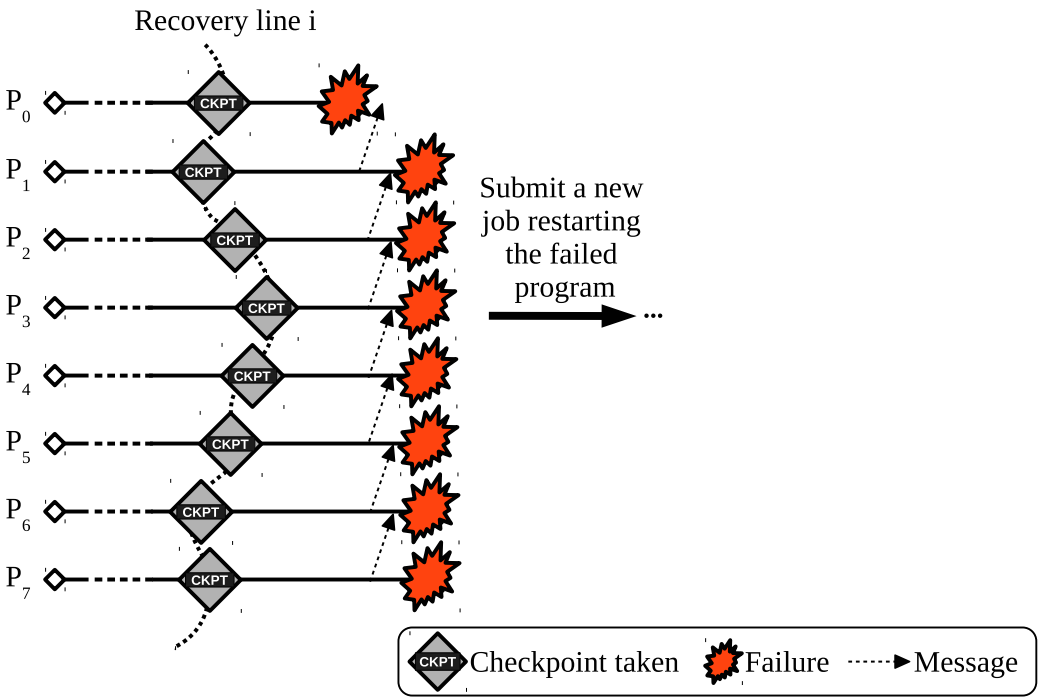
Soluciones stop-and-restart



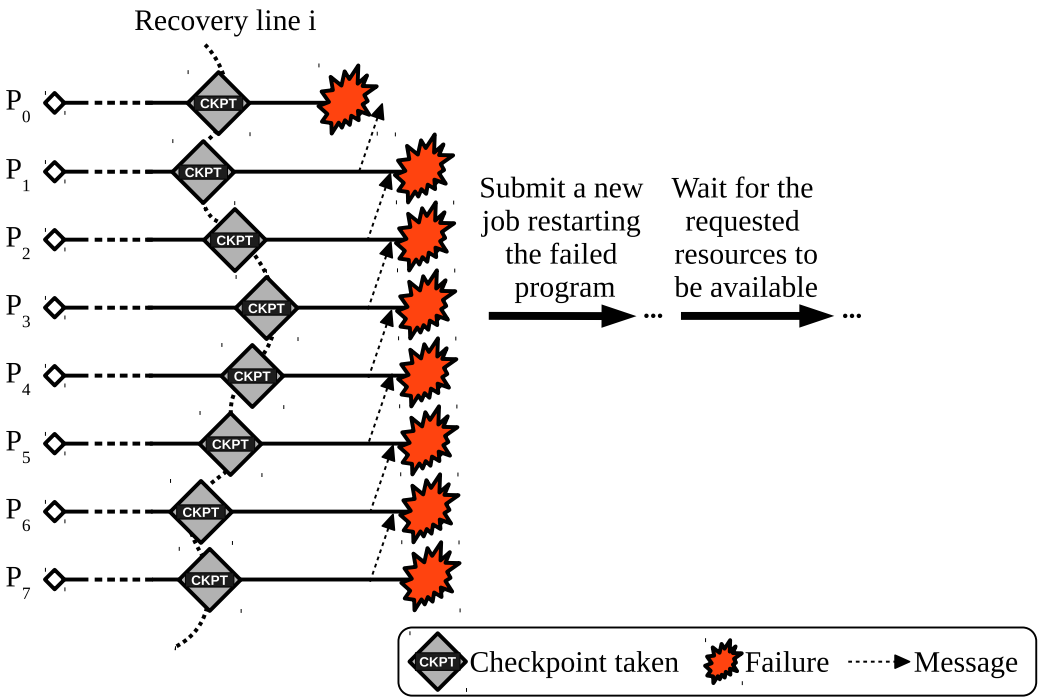
Soluciones stop-and-restart



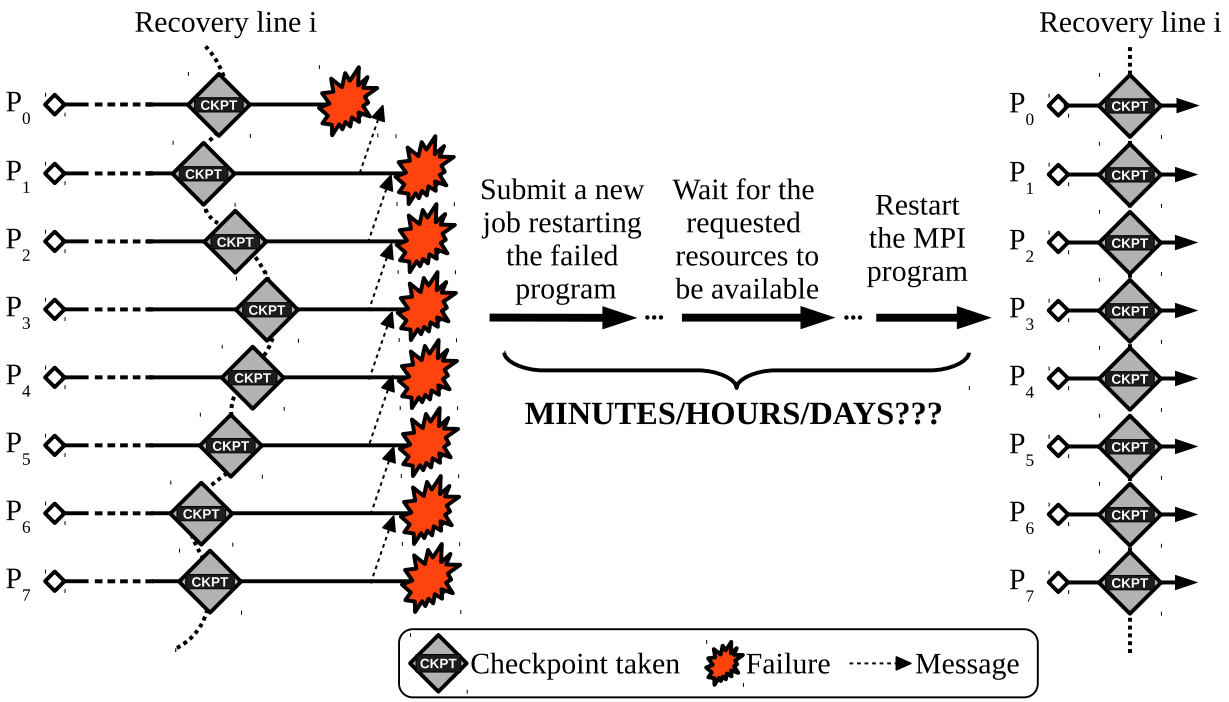
Soluciones stop-and-restart



Soluciones stop-and-restart



Soluciones stop-and-restart



Asignación de diferentes nodos ⇒ contención de red

Aplicaciones resilientes

- Los fallos suelen tener un impacto local
- Un restart completo es innecesario
- Para disminuir la sobrecarga ⇒ aplicaciones MPI resilientes
  - **Son capaces de detectar y reaccionar a fallos sin abortar su ejecución**

User-Level Failure Mitigation (ULFM)

- Propuesto por el Fault Tolerance Working Group del MPI Forum
- No impone un modelo de recuperación concreto
- Incluye nuevas semánticas para:
  - Detectar fallos
  - Propagar fallos
  - Reconfigurar comunicadores

Detección de errores

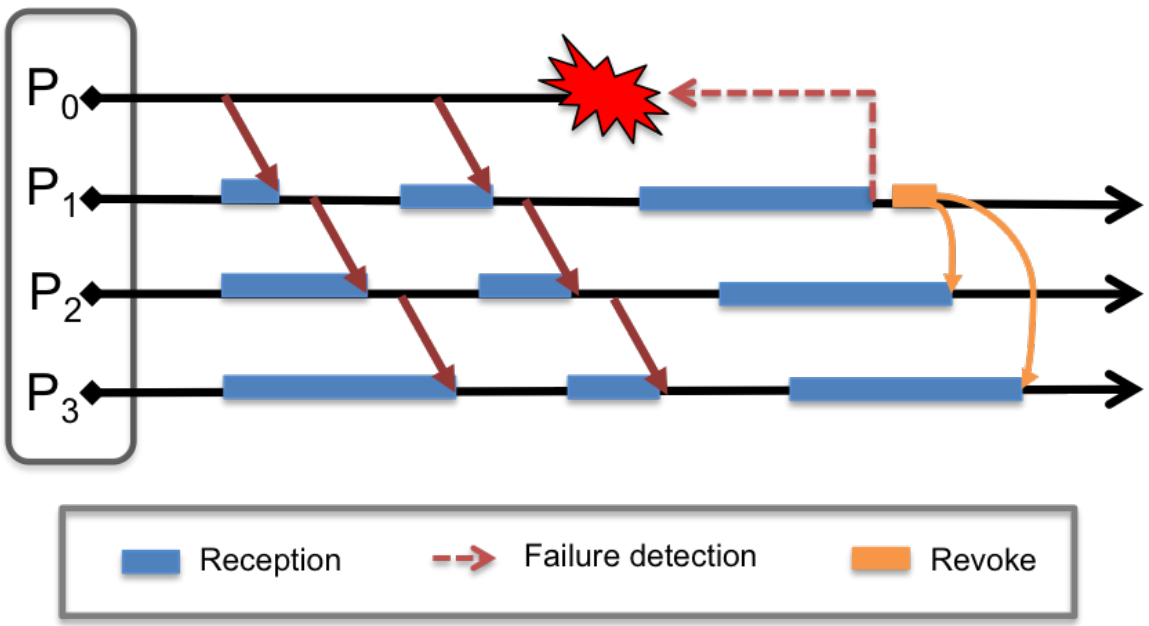
- ULFM define nuevos tipos de errores:
  - MPI_ERR_PROC_FAILED
  - MPI_ERR_PENDING
  - MPI_ERR_REVOKED
- Emplea estructuras y funciones ya presentes en estándar
  - Funciones MPI devuelven código error
  - Los manejadores de error permiten definir comportamiento en caso de error
    - MPI_ERRORS_ARE_FATAL: aborta
    - MPI_ERRORS_RETURN: devuelve código de error

Detección de errores

- Por defecto ULFM solo reporta errores para operaciones cuya semántica no puede ser cumplida
  - Fallos detectados por supervivientes que mantienen comunicaciones con fallidos (MPI_ERR_PROC_FAILED)
- Las operaciones MPI_ANY_SOURCE son interrumpidas con código de error especial (MPI_ERR_PENDING)
- MPIX_Comm_failure_ack & MPIX_Comm_failure_get_acked
  - Permiten hacer acuse de recibo y determinar qué procesos del comunicador han fallado

Propagación de fallos

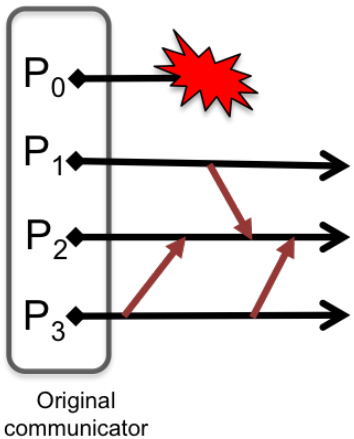
- MPIX_Comm_revoke
  - Propaga el error (MPI_ERR_REVOKED)
  - Invalida el comunicador para futuras comunicaciones



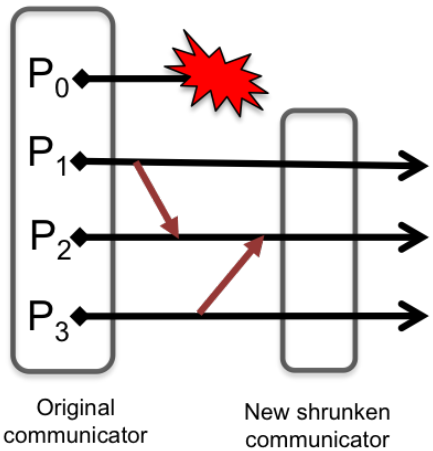
Reconfiguración de los comunicadores

- MPIX_Comm_Shrink: Crea un nuevo comunicador sin los procesos fallidos
  - Restaura la capacidad de hacer comunicaciones colectivas
- Aplicaciones no maleables:
  - MPI_Comm_Spawm genera procesos de reemplazo

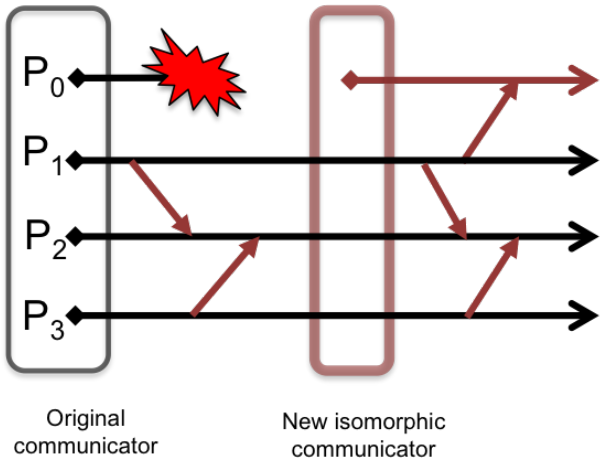
User-level Failure Mitigation (ULFM)



(a) Run-through fault-tolerant applications: point-to-point communication continues uninterrupted between survivors



(b) Malleable applications: communication (including collective operations) can be restored on a reduced set of processes



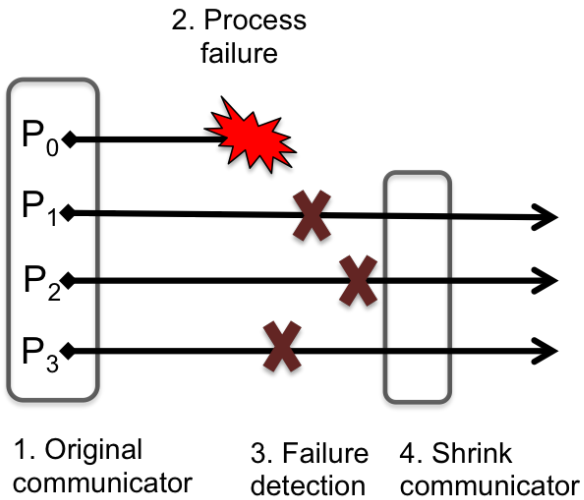
(c) Inflexible applications: isomorphic ranking can be restored either using spare processes, or existing spawn capabilities of MPI

MPIX_Comm_agree

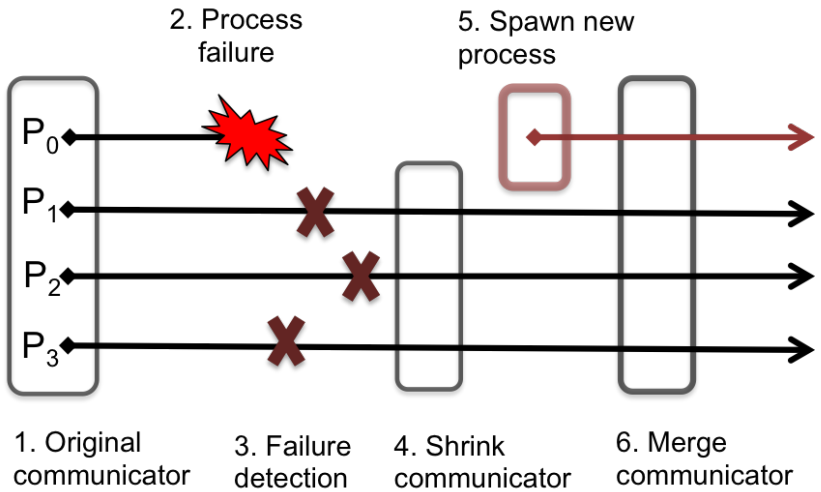
- Operación colectiva que permite consensuar un valor
- Permite la propagación del error de forma más estructurada
- Mete overhead en la ejecución sin fallos

Clasificación:

- **shrinking o non-shrinking**: Reducen o mantienen el número de procesos
- **hacia atrás o hacia adelante**: Se restauran desde un estado previo o intentan encontrar un nuevo estado desde el que continuar
- **local o global**: restaura una parte de la aplicación o la aplicación completa



Shrinking*



Non-shrinking

*Algunos algoritmos iterativos, aplicaciones maestro-esclavo, ...

Inconvenientes solución shrinking

- Impacto negativo en el rendimiento si se pierden varios procesos
- Se puede producir desbalanceo de la carga entre los procesos supervivientes
- No siempre es aplicable

Solución non-shrinking

- Genéricamente aplicable
- Nuevas sobrecargas: transferencia de datos a nuevos procesos

Reincios posibles

- Sin vuelta atrás, con vuelta atrás local, con vuelta atrás global

Inconvenientes solución shrinking

- Impacto negativo en el rendimiento si se pierden varios procesos
- Se puede producir desbalanceo de la carga entre los procesos supervivientes
- No siempre es aplicable

Solución non-shrinking

- Genéricamente aplicable
- Nuevas sobrecargas: transferencia de datos a nuevos procesos

Reincios posibles

- Sin vuelta atrás, con vuelta atrás local, con vuelta atrás global

Inconvenientes solución shrinking

- Impacto negativo en el rendimiento si se pierden varios procesos
- Se puede producir desbalanceo de la carga entre los procesos supervivientes
- No siempre es aplicable

Solución non-shrinking

- Genéricamente aplicable
- Nuevas sobrecargas: transferencia de datos a nuevos procesos

Reincios posibles

- Sin vuelta atrás, con vuelta atrás local, con vuelta atrás global

Detec. fallo	Stop & restart		La aplic. es abortada
	Resiliencia		Notificación fallos a todos los proc. vivos[1]
Reconstruct.	Stop & restart		La aplicación es relanzada
	Resiliencia	Shrinking	Acuerdo sobre los procesos fallidos Se eliminan proc. fallidos del comunic. Rebalanceo de la carga
		Non-shrinking	Acuerdo sobre los proc. fallidos Se eliminan proc. fallidos del comunic. Se generan nuevos procesos (spawn)[2] Se reconstruye el comunicador
Restart	Stop & restart		Recomputación desde el último ckpt
	Resiliencia	Vuelta atrás global	Recomputación desde el último ckpt
		Vuelta atrás local	Recomputación de tareas fallidas
		Sin vuelta atrás	Recomputación no necesaria

[1] En algunos casos es suficiente con notificar a un subconjunto de procesos vivos
[2] O activación de procesos de repuesto ya existentes

Detec. fallo	Stop & restart		La aplic. es abortada
	Resiliencia		Notificación fallos a todos los proc. vivos[1]
Reconstruct.	Stop & restart		La aplicación es relanzada
	Resiliencia	Shrinking	Acuerdo sobre los procesos fallidos Se eliminan proc. fallidos del comunic. Rebalanceo de la carga
		Non-shrinking	Acuerdo sobre los proc. fallidos Se eliminan proc. fallidos del comunic. Se generan nuevos procesos (spawn)[2] Se reconstruye el comunicador
Restart	Stop & restart		Recomputación desde el último ckpt
	Resiliencia	Vuelta atrás global	Recomputación desde el último ckpt
		Vuelta atrás local	Recomputación de tareas fallidas
		Sin vuelta atrás	Recomputación no necesaria

[1] En algunos casos es suficiente con notificar a un subconjunto de procesos vivos
[2] O activación de procesos de repuesto ya existentes

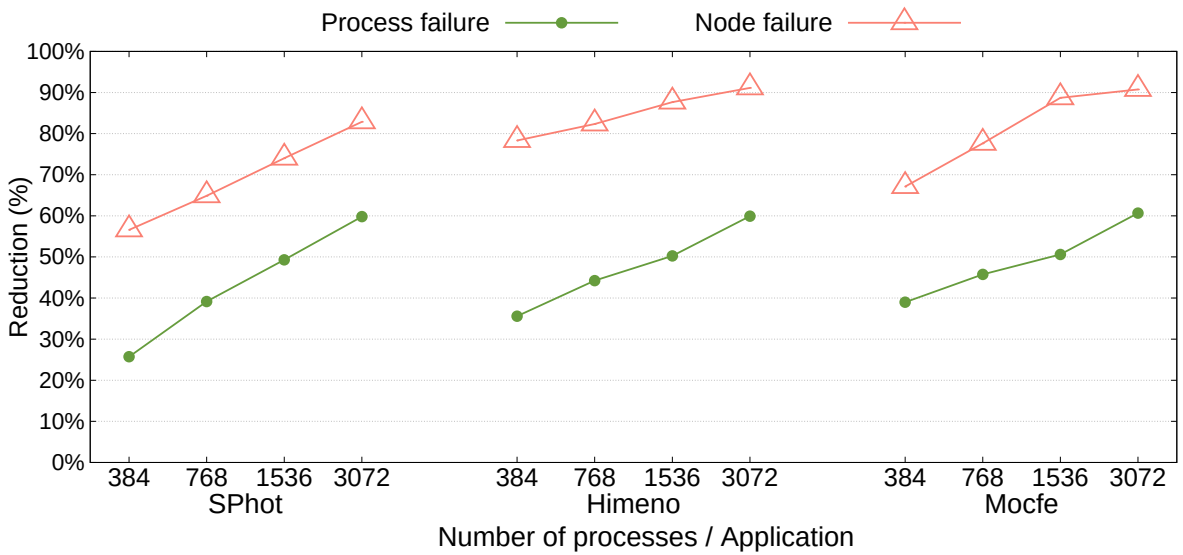
Evaluación

- Tiempos detección Stop-and-restart vs resiliencia

Sistema & Aplicaciones

- Supercomputador FinisTerrae-II del CESGA
  - Nodos: 2 $\times$ Intel Xeon E5-2680 $\rightarrow$ 24 núcleos & 128GB RAM
  - Red: InfiniBand FDR 56Gb/s
- 3 aplicaciones:
  - Himeno: resolutor ecuación Poisson
  - Mocfe: simula procedimientos principales de código de método de características 3D
  - Sphot: transporte de fotones 2D (benchmarks ASC Sequoia)
- 1 proceso MPI por núcleo

N. Losada, M.J. Martín and P. González . *Assessing resilient versus stop-and-restart fault-tolerant solutions in MPI applications.*The Journal of Supercomputing 73, 1 (2017), 316–329.



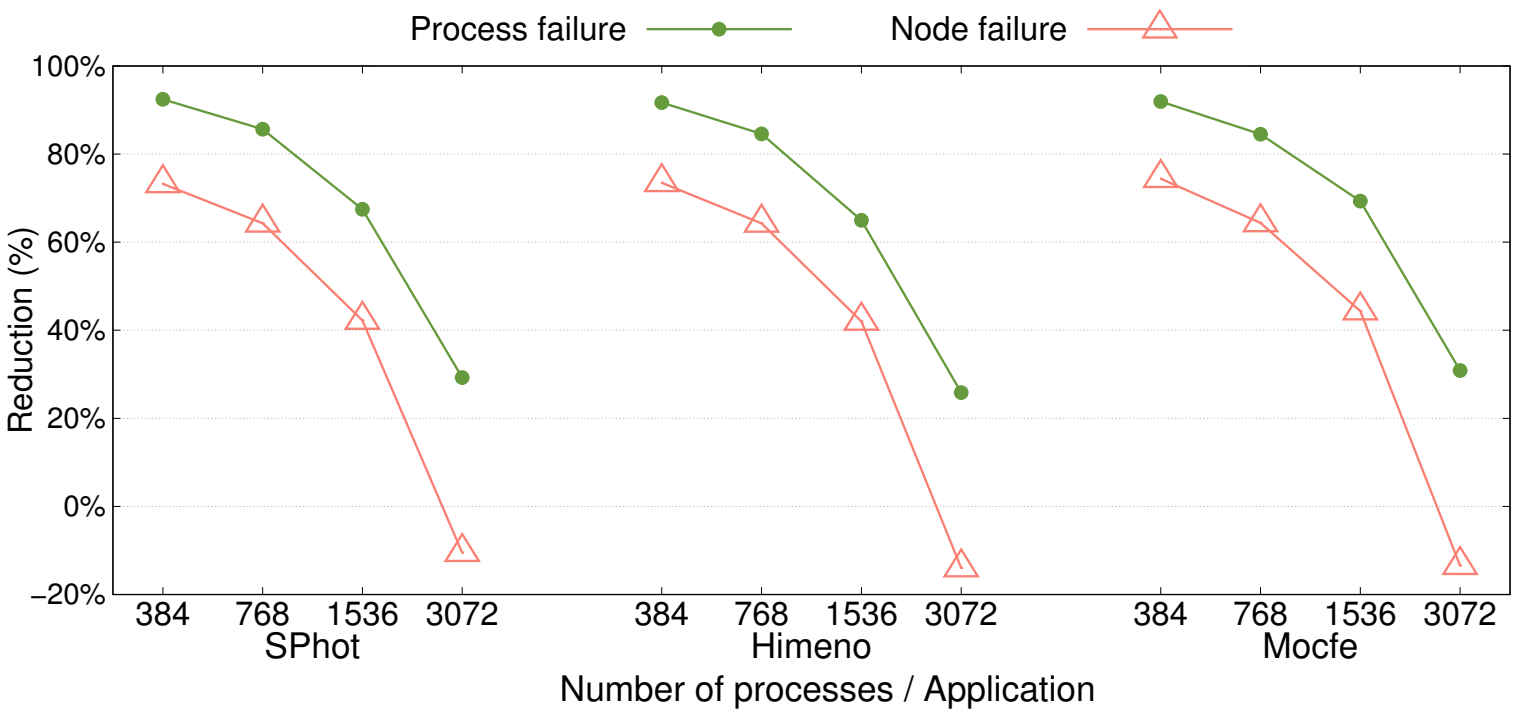
Reducción media cuando falla un proceso: 47%
Reducción media cuando falla un nodo: 79%

Detec. fallo	Stop & restart		La aplic. es abortada
	Resiliencia		Notificación fallos a todos los proc. vivos[1]
Reconstruct.	Stop & restart		La aplicación es relanzada
	Resiliencia	Shrinking	Acuerdo sobre los procesos fallidos Se eliminan proc. fallidos del comunic. Rebalanceo de la carga
		Non-shrinking	Acuerdo sobre los proc. fallidos Se eliminan proc. fallidos del comunic. Se generan nuevos procesos (spawn)[2] Se reconstruye el comunicador
Restart	Stop & restart		Recomputación desde el último ckpt
	Resiliencia	Vuelta atrás global	Recomputación desde el último ckpt
		Vuelta atrás local	Recomputación de tareas fallidas
		Sin vuelta atrás	Recomputación no necesaria

[1] En algunos casos es suficiente con notificar a un subconjunto de procesos vivos
[2] O activación de procesos de repuesto ya existentes

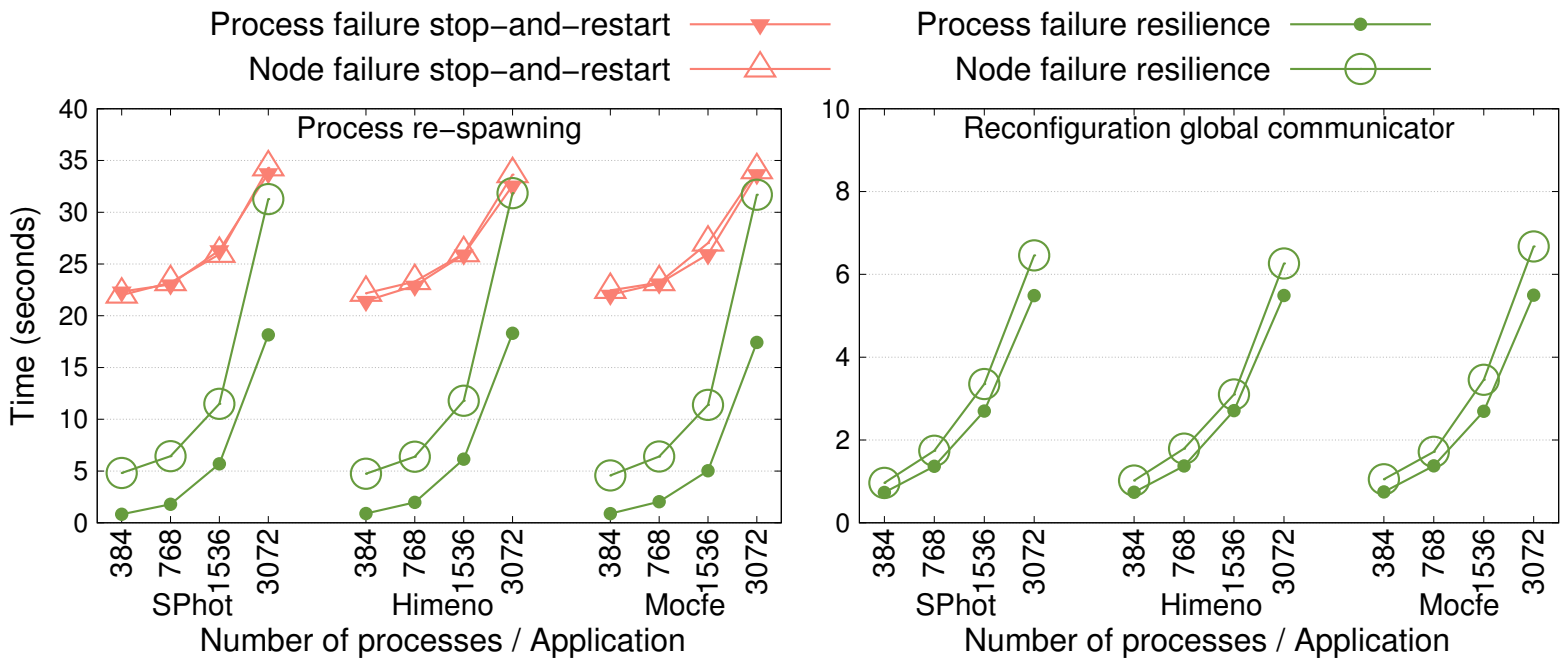
Reducción tiempo operaciones reconstrucción con ULFM

Aplicación MPI resiliente non-shrinking vs stop-and-restart



Tiempo operaciones más costosas de la reconstrucción

Spawn de nuevos procesos y reconstrucción comunicadores



Detec. fallo	Stop & restart		La aplic. es abortada
	Resiliencia		Notificación fallos a todos los proc. vivos[1]
Reconstruct.	Stop & restart		La aplicación es relanzada
	Resiliencia	Shrinking	Acuerdo sobre los procesos fallidos Se eliminan proc. fallidos del comunic. Rebalanceo de la carga
		Non-shrinking	Acuerdo sobre los proc. fallidos Se eliminan proc. fallidos del comunic. Se generan nuevos procesos (spawn)[2] Se reconstruye el comunicador
Restart	Stop & restart		Recomputación desde el último ckpt
	Resiliencia	Vuelta atrás global	Recomputación desde el último ckpt
		Vuelta atrás local	Recomputación de tareas fallidas
		Sin vuelta atrás	Recomputación no necesaria

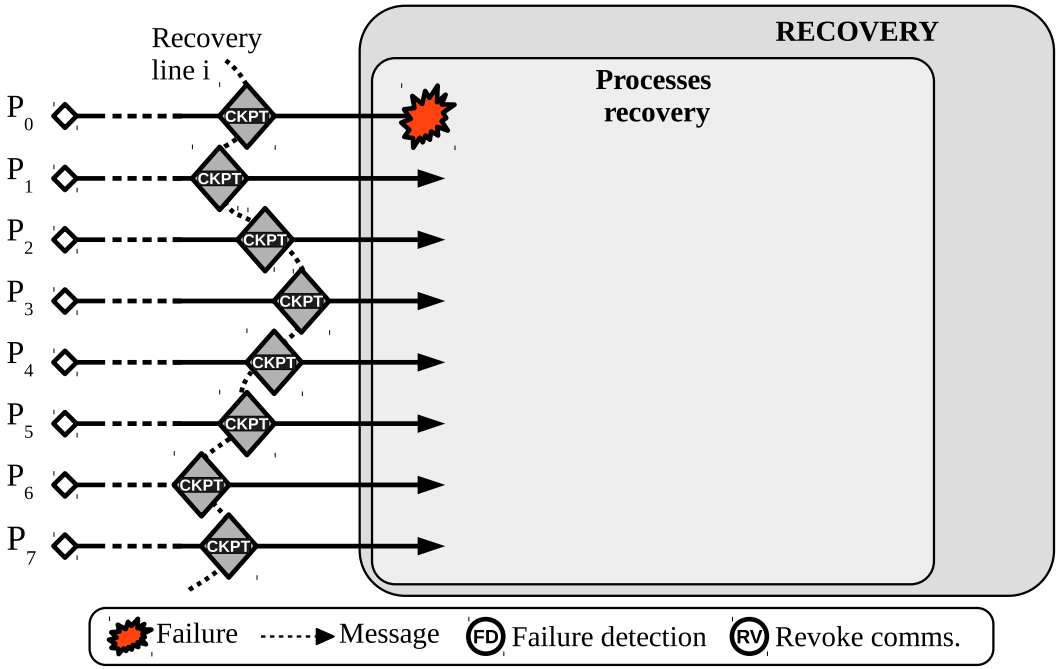
[1] En algunos casos es suficiente con notificar a un subconjunto de procesos vivos
[2] O activación de procesos de repuesto ya existentes

Detec. fallo	Stop & restart		La aplic. es abortada
	Resiliencia		Notificación fallos a todos los proc. vivos[1]
Reconstruct.	Stop & restart		La aplicación es relanzada
	Resiliencia	Shrinking	Acuerdo sobre los procesos fallidos Se eliminan proc. fallidos del comunic. Rebalanceo de la carga
		Non-shrinking	Acuerdo sobre los proc. fallidos Se eliminan proc. fallidos del comunic. Se generan nuevos procesos (spawn)[2] Se reconstruye el comunicador
Restart	Stop & restart		Recomputación desde el último ckpt
	Resiliencia	Vuelta atrás global	Recomputación desde el último ckpt
		Vuelta atrás local	Recomputación de tareas fallidas
		Sin vuelta atrás	Recomputación no necesaria

[1] En algunos casos es suficiente con notificar a un subconjunto de procesos vivos
[2] O activación de procesos de repuesto ya existentes

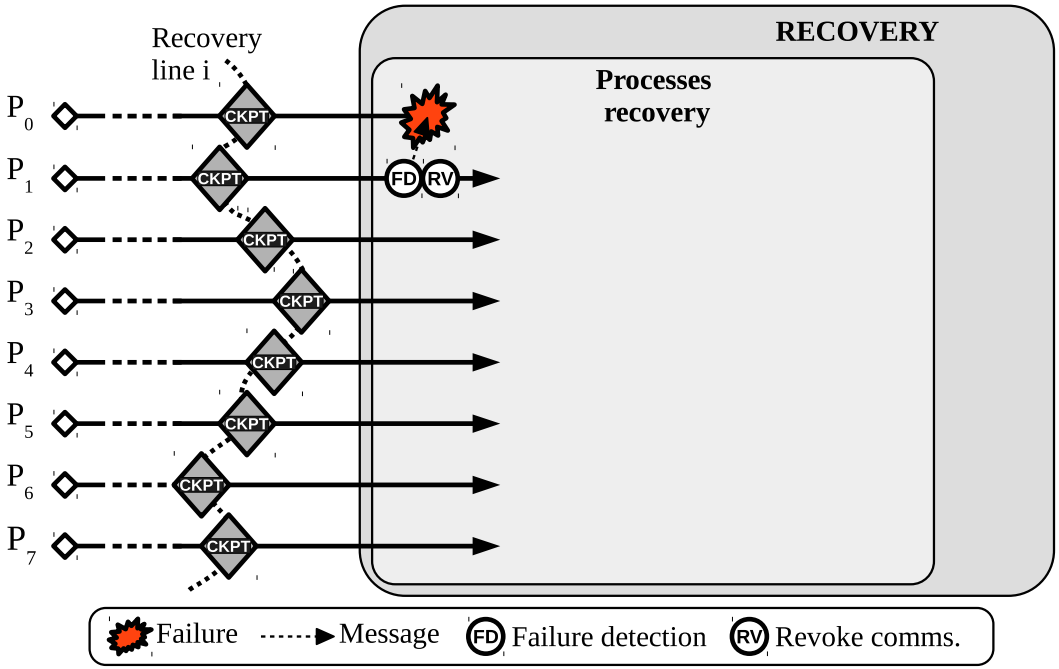
Nuestra propuesta

- Vuelta atrás local para aplicaciones MPI SPMD genéricas
  - Sólo los procesos afectados por el fallo van atrás
- Combina ULFM, CPPC y el logging de mensajes
- Colaboración con Innovative Computing Laboratory (Univ. Tennessee)



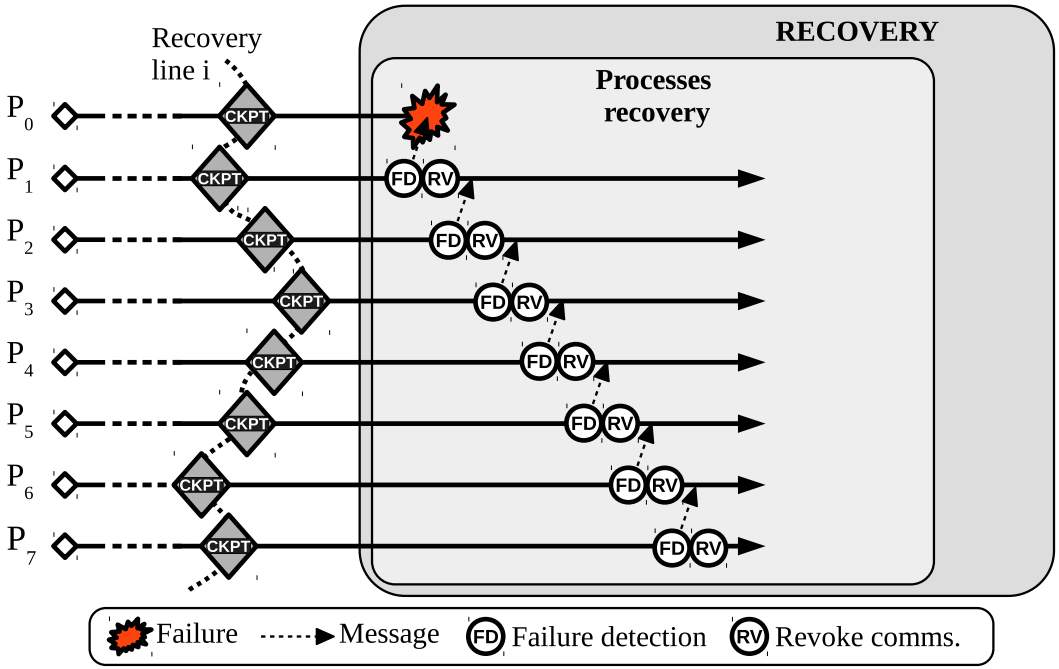
Primer paso: recuperación de los procesos

- Se utilizan las funcionalidades de ULFM



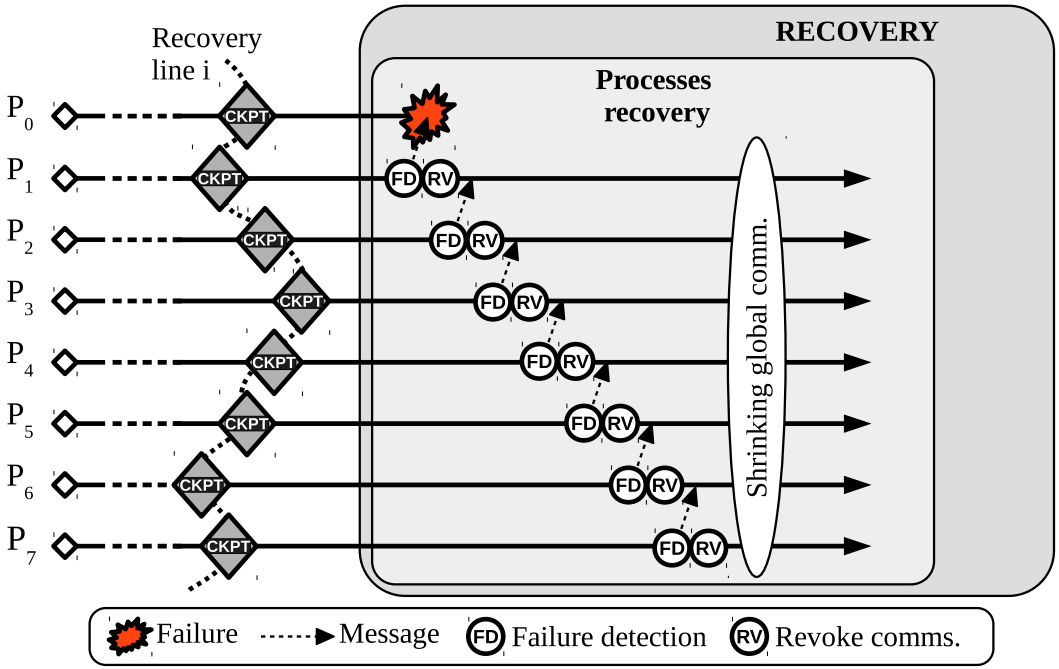
Primer paso: recuperación de los procesos

- Los supervivientes que se comunican con un fallido detectan el fallo y revocan comunicadores



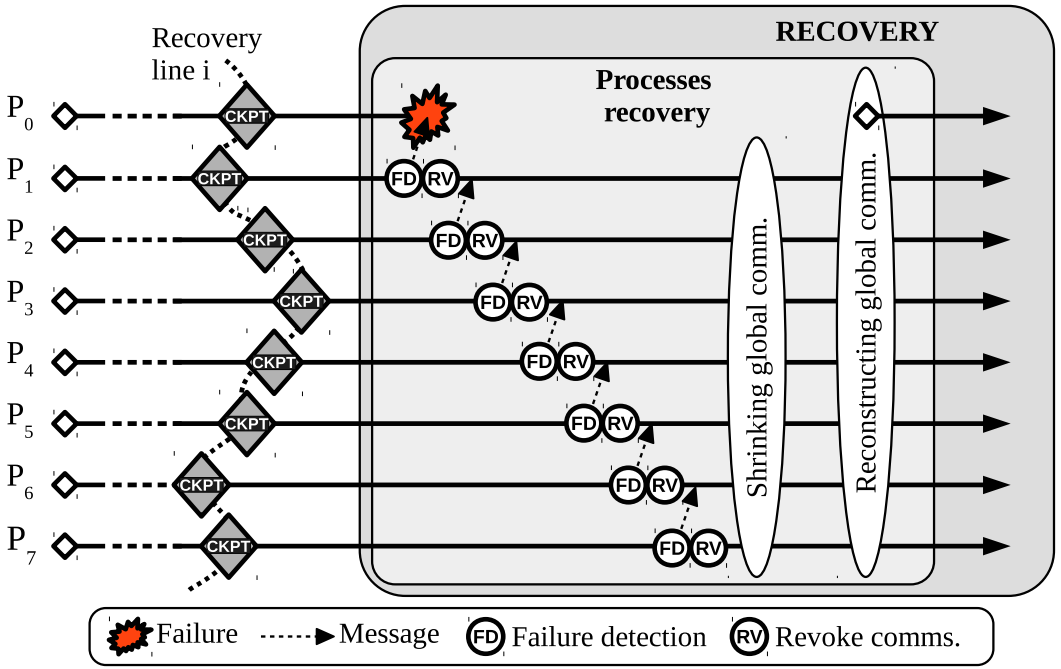
Primer paso: recuperación de los procesos

- El error se propaga a todos los supervivientes



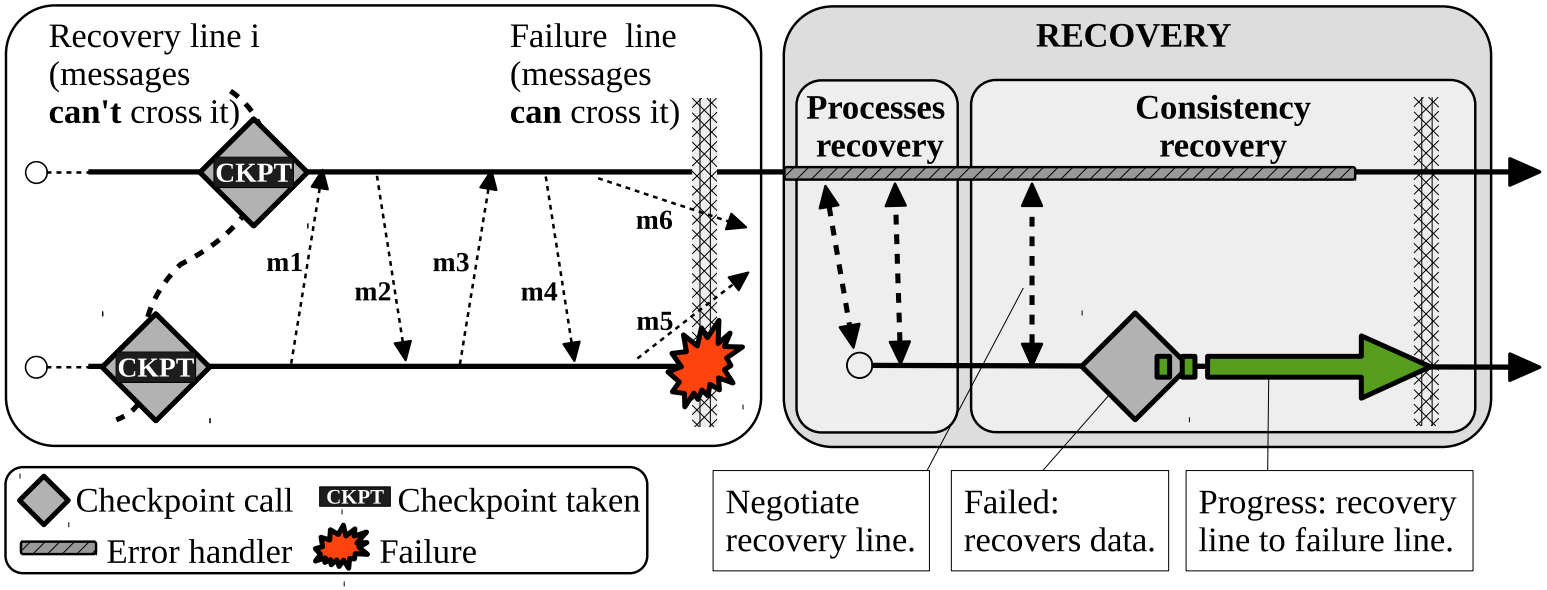
Primer paso: recuperación de los procesos

- Supervivientes acuerdan procesos fallidos
- Excluyen procesos fallidos del comunicador



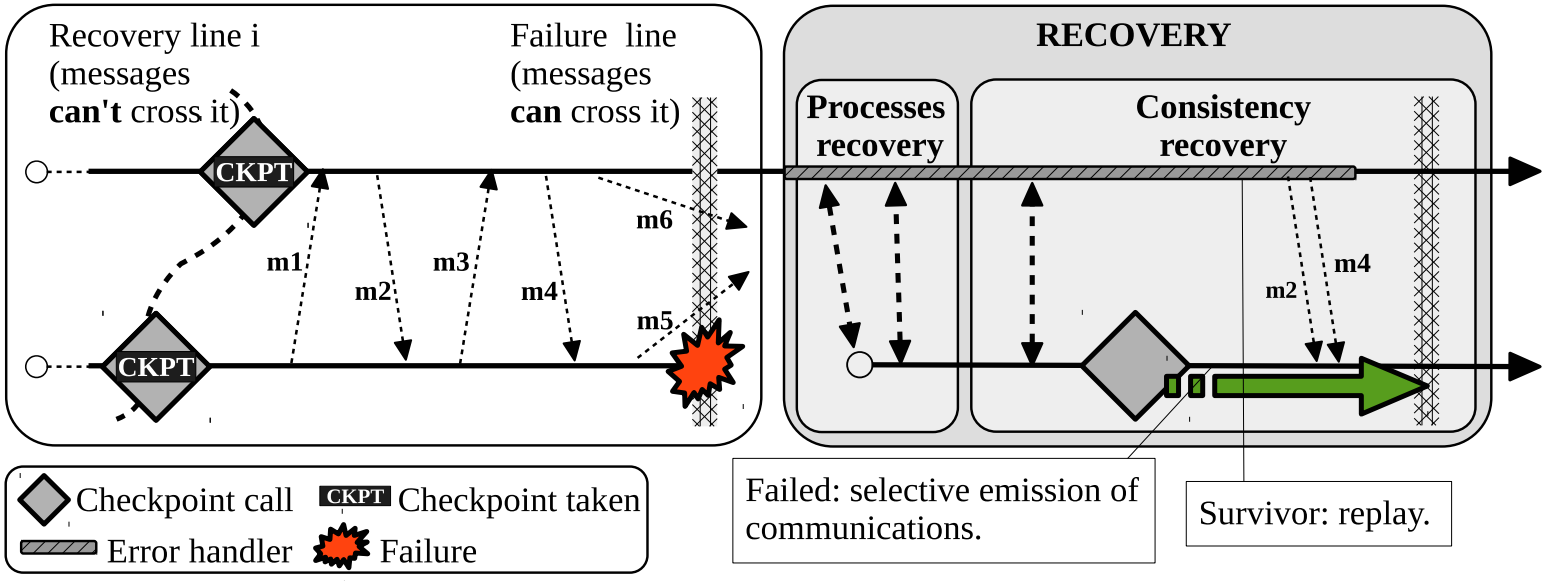
Primer paso: recuperación de los procesos

- Se lanzan nuevos procesos para sustituir fallidos
- Se reconstruye el comunicador



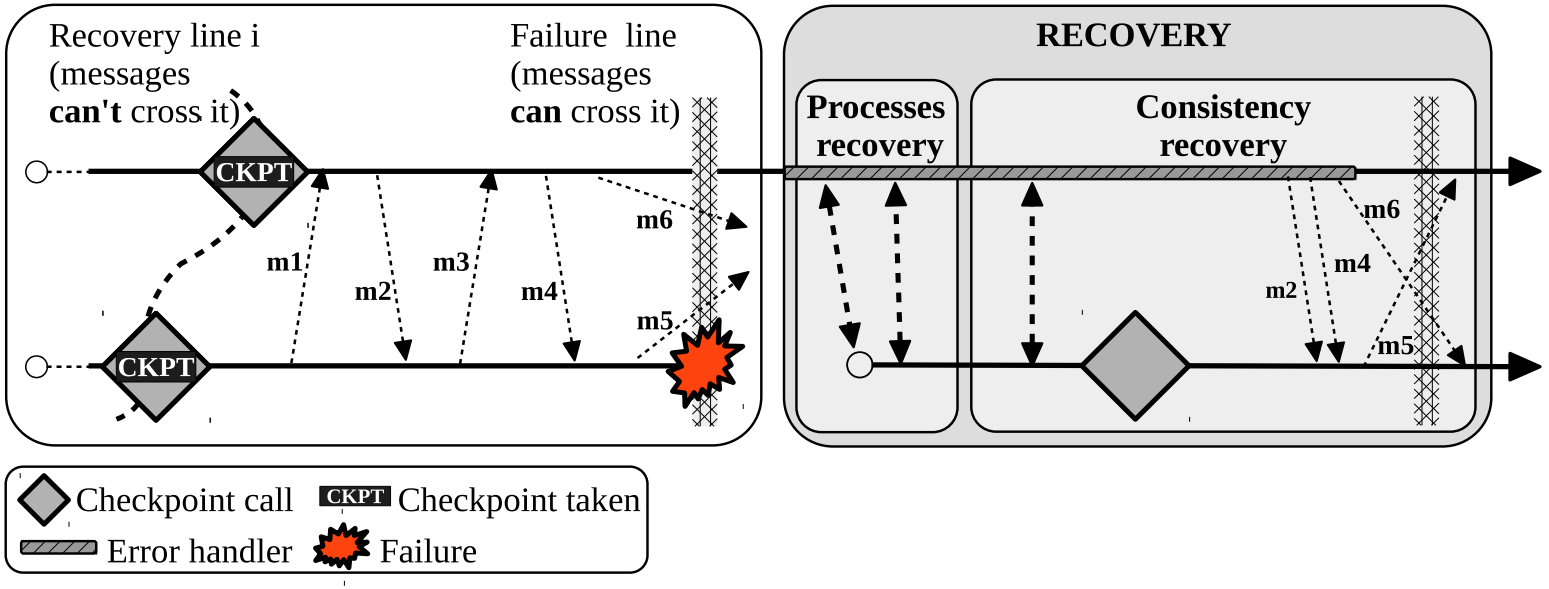
Segundo paso: Recuperación de la consistencia

- Se negocia la linea de recuperación
- Procesos fallidos:
  - Recuperan los datos del fichero de checkpointing
  - Avanzan desde la línea de recuperación a la línea de fallos



Segundo paso - Progreso procesos fallidos

- Para llegar al mismo estado:
  - Algunos mensajes tienen que ser enviados de nuevo
  - Otros tienen que ser saltados



Segundo paso - Consistencia supervivientes

- Hay que reenvíar también las comunicaciones interrumpidas por los fallos

Logging de los mensajes

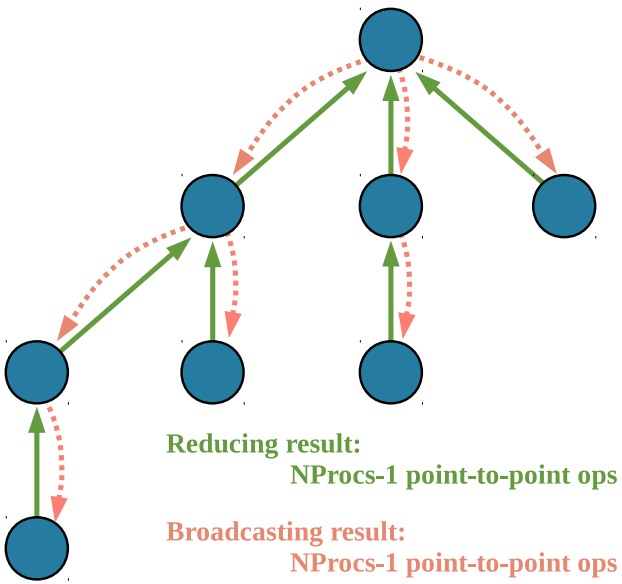
- Logging a nivel de sistema para comunicaciones punto a punto (VProtocol)
- Logging a nivel de aplicación para comunicaciones colectivas

Seguimiento de los mensajes

- Utilizamos los SSIDs (Sender Sequence IDs) de los mensajes

Reducción tamaño logs

- Protocolo de coordinación espacial de CPPC:
  - Comunicaciones no pueden cruzar línea recuperación $\Rightarrow$ Logs pueden ser borrados en líneas recuperación ($\downarrow$ uso memoria).
- Comunicaciones colectivas - Log a nivel de aplicación:
  - No hace log de comunicaciones punto-a-punto internas
  - Reduce tamaño log en colectivas con copias intermedias



Ejemplo: MPI_Allreduce con árbol binomial

- Log comunicaciones punto-a-punto:
  $2 \times (Nprocs - 1) \times Buffsize$ bytes
- Log a nivel de aplicación:
  $Nprocs \times Buffsize$ bytes

Sistema & Applicaciones

- Supercomputador FinisTerrae-II del CESGA.
  - Nodos: 2 $\times$ Intel Xeon E5-2680 $\rightarrow$ 24 núcleos & 128GB RAM
  - Red: InfiniBand FDR 56Gb/s
- Checkpoints almacenados en disco (Lustre sobre InfiniBand)
- 4 aplicaciones: Himeno, Mocfe, Sphot, Tealeaf
  - Tealeaf: ecuación lineal conducción calor (proyecto Mantevo)
  - 1 proceso MPI por núcleo

Tiempos de ejecución originales en minutos

	48P	96P	192P	384P	768P
HIMENO	18.48	9.22	4.76	2.45	1.56
MOCFE	20.49	8.26	4.75	2.41	1.48
SPHOT	16.38	8.25	3.78	2.25	1.48
TEALEAF	17.50	9.31	4.28	2.35	1.79

Pruebas

- Comparación vuelta atrás local vs. vuelta atrás global
  - Ambas implementan resiliencia usando ULFM
  - Checkpointing al 40% de la ejecución

Tiempos de ejecución originales en minutos

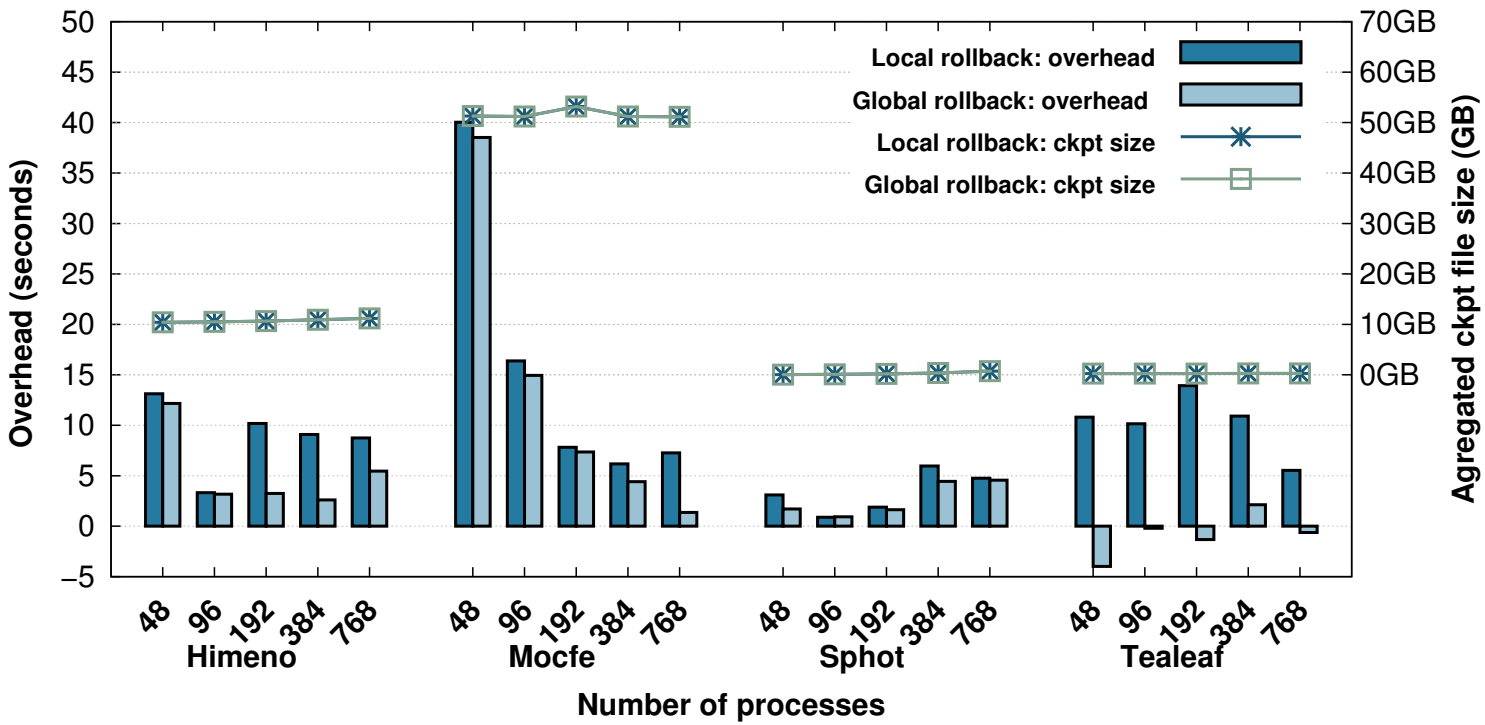
	48P	96P	192P	384P	768P
HIMENO	18.48	9.22	4.76	2.45	1.56
MOCFE	20.49	8.26	4.75	2.41	1.48
SPHOT	16.38	8.25	3.78	2.25	1.48
TEALEAF	17.50	9.31	4.28	2.35	1.79

Pruebas

- Comparación vuelta atrás local vs. vuelta atrás global
  - Ambas implementan resiliencia usando ULFM
  - Checkpointing al 40% de la ejecución

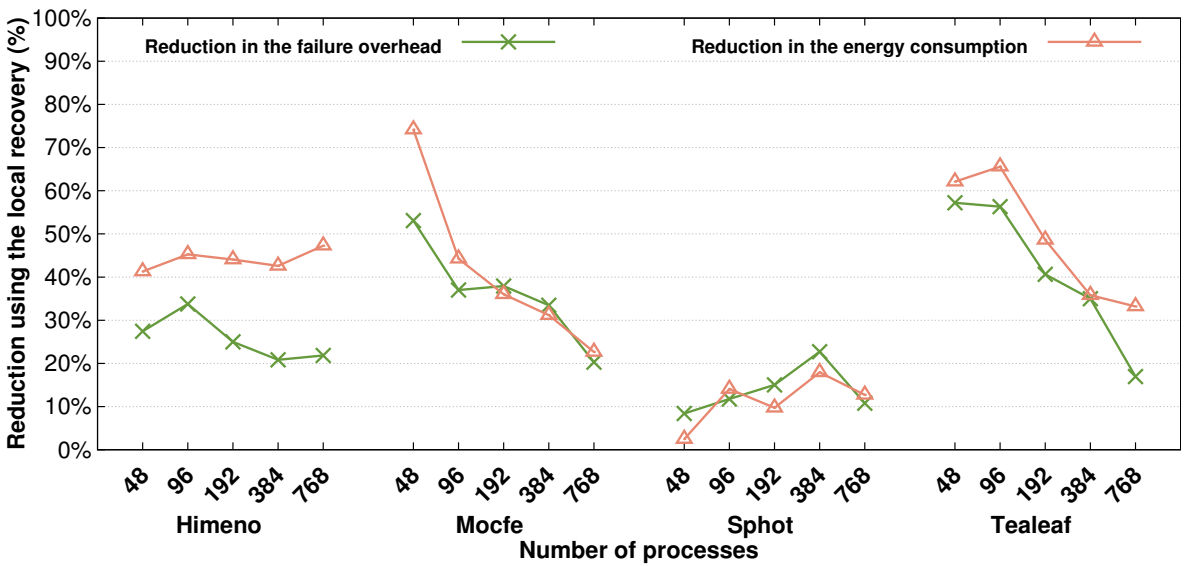
Overhead en la ejecución sin fallos

Vuelta atrás local (ckpt+log) vs Vuelta atrás global (ckpt)



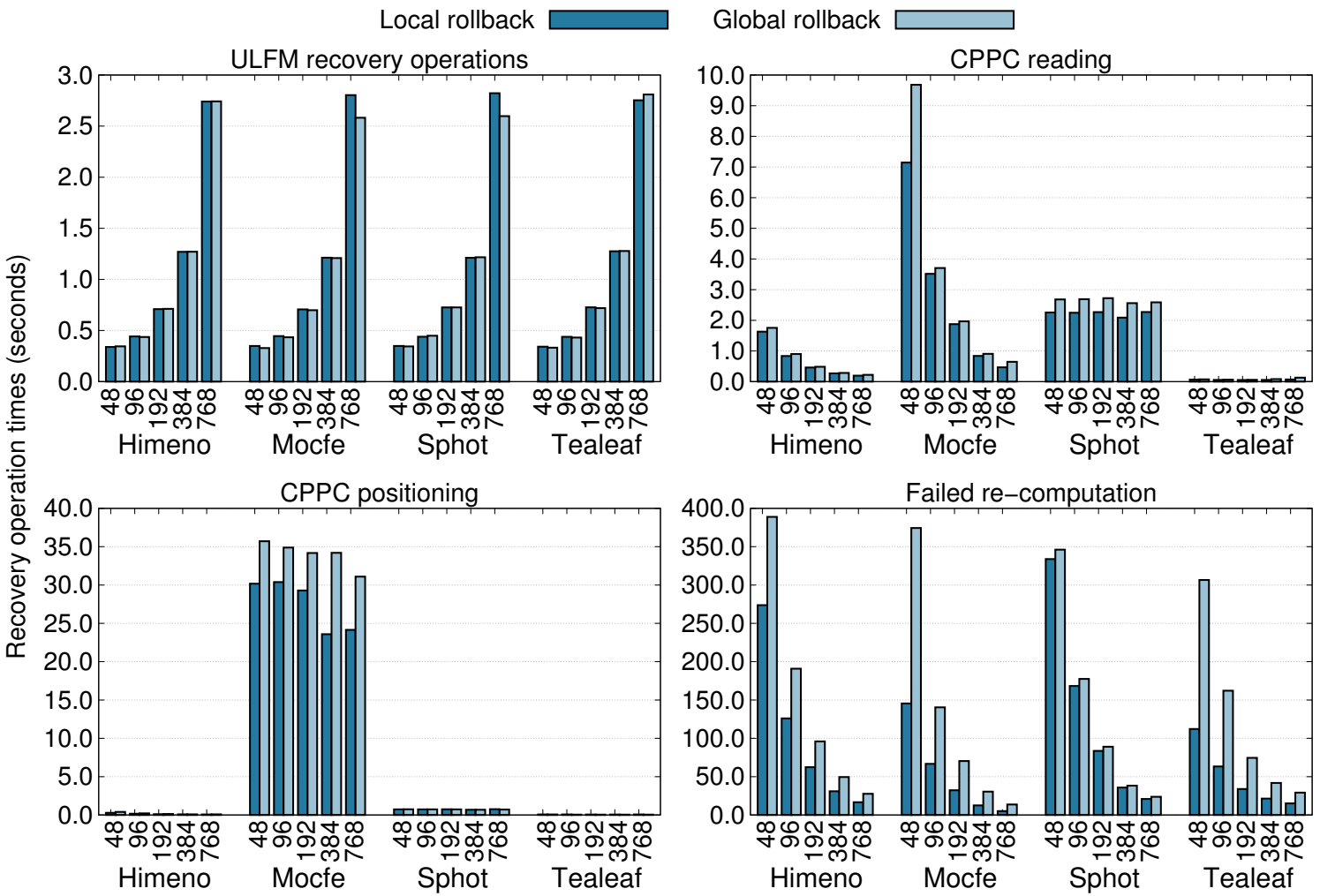
Overhead en la ejecución con fallos

- Fallo al 75% de la ejecución (matando el último proceso).
- Reducción (cuando se usa vuelta atrás local vs global) en:
  - Tiempo de ejecución extra y energía consumida



Reducción tiempo extra: 30% en media

Evaluación experimental



1 Introducción

2 Soluciones *Stop-and-restart*

3 Aplicaciones MPI Resilientes

4 **Conclusiones**
- Conclusiones

Conclusiones

- ↑ Núm. procesadores ⇒ ↓ MTTF
- Soluciones tradicionales:
  - Stop-and-restart⇒ ↑ overhead
- Alternativa:
  - Aplicaciones resilientes
- ULFM funciones básicas de bajo nivel para construir aplicaciones resilientes ⇒ No trivial
- Se necesitan librerías más alto nivel para facilitar desarrollo aplicaciones resilientes

Nuevas tendencias en tolerancia a fallos para aplicaciones paralelas

María J. Martín

Grupo de Arquitectura de Computadores - CITIC
Universidade da Coruña
mariam@udc.es