UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

FACTORIZACIÓN LDL

Rendimiento MPI en un proveedor de Coud Computing

Mihaita Alexandru Lupoiu

8 Enero 2016

Conceptos de la Computación en Grid Y Cloud

TRABAJO PROPUESTO

1. Introdución
2. Ejemplo
3. Entorno
4. Configuración
5. Código MPI
6. Resultados
7. Conclusion y Trabajos Futuros
8. Resumen

INTRODUCIÓN

INTRODUCIÓN LDL'

La factorización LDL' es una forma de factorización de una matriz A como el producto de una matriz triangular inferior L por una diagonal D y por una matriz inferior traspuesta L'.

$$A = L * D * L^T$$

Para evitar conplicaciones las pruebas se harán solo para matrices simétricas, eso significa que A = A'.

EJEMPLO

EJEMPLO

Ejemplo sin sobre-escritura:

$$A = \begin{bmatrix} 4 & 3 & 1 & 1 \\ 3 & 8 & 1 & 2 \\ 1 & 1 & 16 & 1 \\ 1 & 2 & 1 & 10 \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.75 & 1 & 0 & 0 \\ 0.25 & 0.0435 & 1 & 0 \\ 0.25 & 0.2174 & 0.0442 & 1 \end{bmatrix}}_{L} \times \underbrace{\begin{bmatrix} 4 & & & \\ & 5,75 & & \\ & & 15.7391 & \\ & & & 9.4475 \end{bmatrix}}_{D} \times \underbrace{\begin{bmatrix} 1 & 0.75 & 0.25 & 0.25 \\ 0 & 1 & 0.0435 & 0.2174 \\ 0 & 0 & 1 & 0.0442 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{L'}$$

EJEMPLO

Ejemplo con sobre-escritura:

$$A = \begin{bmatrix} 4 & 3 & 1 & 1 \\ 3 & 8 & 1 & 2 \\ 1 & 1 & 16 & 1 \\ 1 & 2 & 1 & 10 \end{bmatrix}$$

$$LDL' = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0.75 & 5,75 & 0 & 0 \\ 0.25 & 0.0435 & 15.7391 & 0 \\ 0.25 & 0.2174 & 0.0442 & 9.4475 \end{bmatrix}$$

ENTORNO

MICROSOFT AZURE

Microsoft Azure es una plataforma general que tiene diferentes servicios y alojada en los Data Centers de Microsoft.

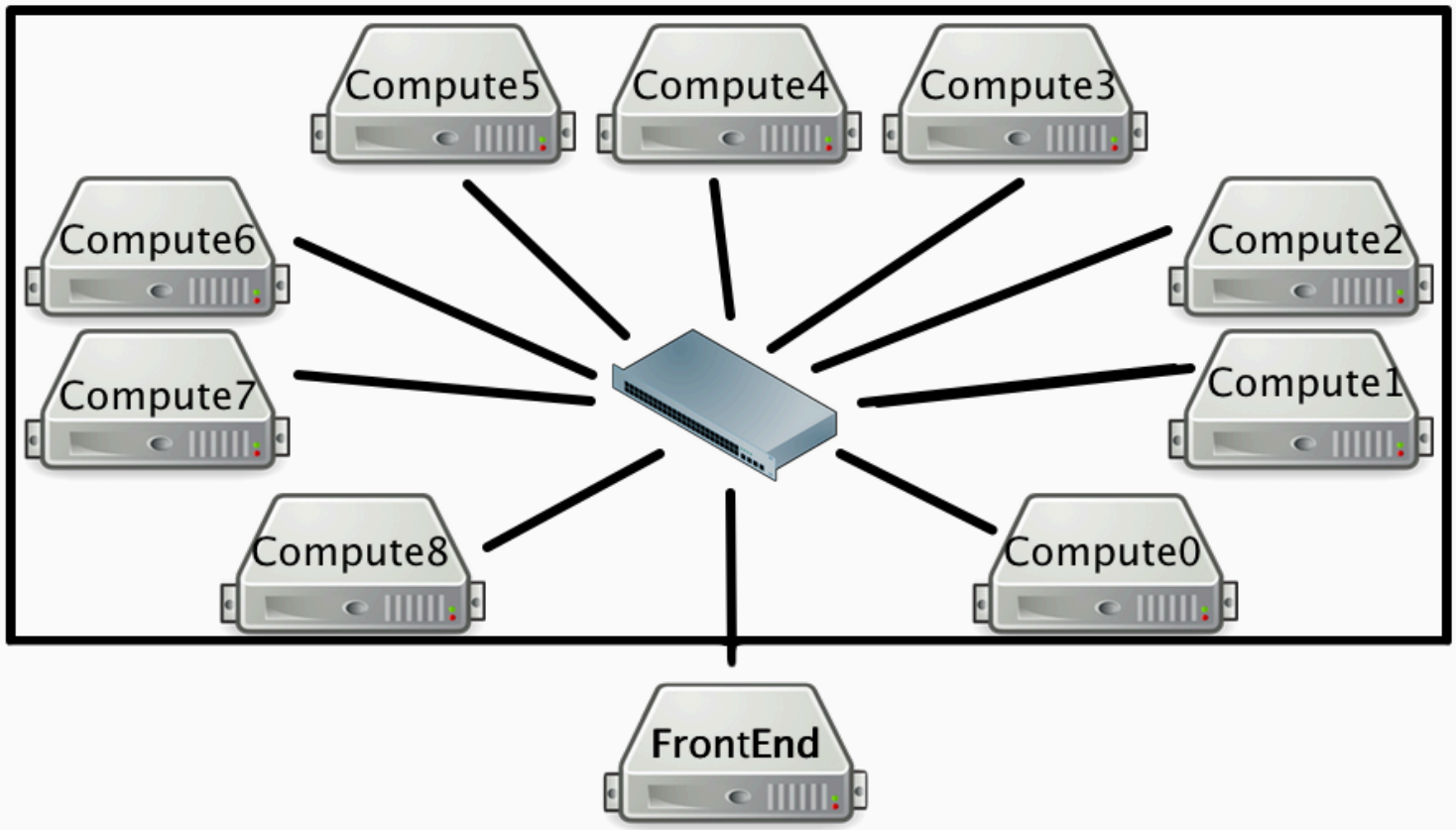
El servicio que más interesa para la realización de este proyecto es el modelo de Infraestructura como servicio. Azure proporciona máquinas preparadas para funcionar con MPI sobre una red Infiniband, pero a un precio muy elevado (alrededor de 4€ la hora).

A la hora de configurar el MPI en las instancias convencionales hay que tener en cuenta los siguientes aspectos:

1. Limitaciones de los Puertos públicos.
2. Máxima proximidad.
3. Configuración apropiada.

ESTRUCTURA NODOS AZURE

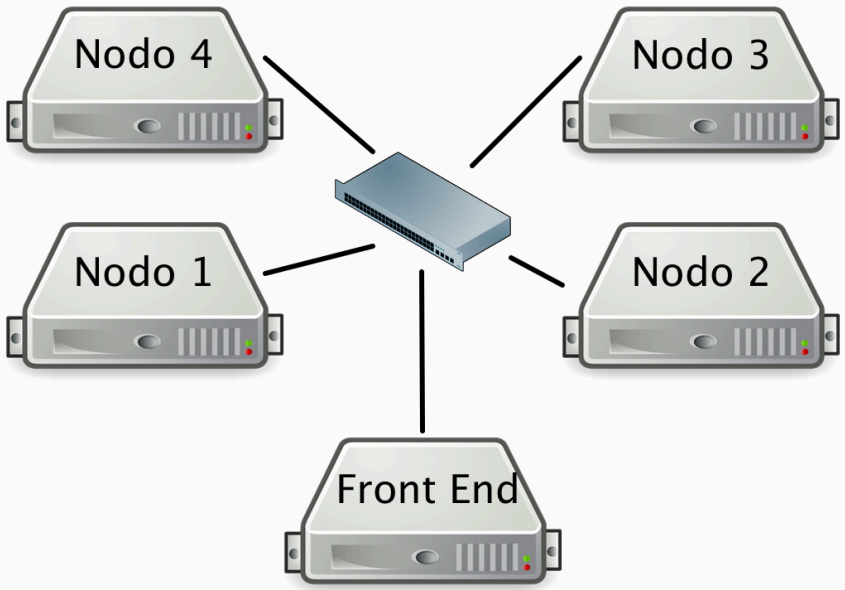
La estructura que se pretende tener es la siguiente:



QUADCLUSTER UPV

Quadcluster es un cluster que pertenece al departamento Sistemas Informáticos y Computación de la Universidad DSIC.

Está formada un nodo de comunicación y 4 nodos de computo conectados mediante una conexion de 1 Gb Ethernet, con una CPU Intel(R) Xeon(R) CPU X5365 @ 3.00GHz de 8 Núcleos y 16 GB de RAM DDR2.



CONFIGURACIÓN

CONFIGURACIÓN

Se tiene que configurar solo el entorno de Windows Azure ya que en Quadcluster está todo preparado para el uso de MPI.

Los pasos realizados en Azure son los siguientes:

1. Se ha creado una máquina FrontEnd con su conjunto de disponibilidad.
2. Se ha instalado también los binarios de openmpi y el entorno de desarrollo.
3. Se ha creado 19 máquinas compute dentro del conjunto de disponibilidad.
4. Se ha configurado las máquinas compute para que no requiera introducir la contraseña mediante el uso de un certificado x.509.
5. Se ha instalado solo los binarios de openmpi.

CONFIGURACIÓN

Para la máquina FrontEnd, se ha creado el certificado con el comando:

```
ssh-keygen -t rsa
```

Esto generara la siguiente salida:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/azureuser/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/azureuser/.ssh/id_rsa.
Your public key has been saved in /home/azureuser/.ssh/id_rsa.pub.
The key fingerprint is:
59:8a:98:37:33:eb:68:52:c1:bc:88:d0:fd:87:39:00 azureuser@frontend
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|  E              |
| . =      .      |
|. . *o . +       |
|.. .o=*oS        |
|. . o.==.        |
|   .  .o         |
|  . .o           |
|   o. .          |
+-----------------+
```

CONFIGURACIÓN

Para la creación de las 19 máquinas de computo, se ha querido automatizar el proceso de difundir la clave publica y de instalación de los binarios de openmpi.

Para ello se ha creado el siguiente script:

```
1  #!/usr/bin/env bash
2  if [ -z "$1" ]
3  then
4      echo "Start Argument not supplied!";
5      exit;
6  fi
7  if [ -z "$2" ]
8  then
9      echo "End Argument not supplied!";
10     exit;
11 fi
12 for (( i=$1; i <= $2; i=i+1 ))
13 do
14 ^^Iecho "--------------------------------"
15 ^^Iecho "Sending id.rsa.pub to compute$i"
16 ^^Iecho "--------------------------------"
17 ^^Icat /home/azureuser/.ssh/id_rsa.pub | ssh azureuser@compute$i 'cat >> .ssh/authorized_keys'
18
19 ^^Issh azureuser@compute$i 'sudo apt-get update'
20 ^^Issh azureuser@compute$i 'sudo apt-get -y install openmpi-bin'
21 done
```

CONFIGURACIÓN

Resultado Final:

NOMBRE	ESTADO	SUSCRIPCIÓN	UBICACIÓN	NOMBRE DE DNS
compute0	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute1	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute10	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute11	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute12	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute13	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute14	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute15	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute16	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute17	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute18	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute2	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute3	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute4	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute5	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute6	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute7	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute8	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
compute9	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net
frontend	En ejecución	Pase para Azure	Norte de Europa	frontendmihai.cloudapp.net

CÓDIGO MPI

CÓDIGO MPI: REPARTO DATOS MÁSTER

```
1     if (pid == root){
2         time_start=MPI_Wtime();
3         int s = matrix_size/nprocs;
4         int i,j,k;
5         for (i = 1; i < nprocs; ++i) {
6             MPI_Send(&s, 1, MPI_INT, i, 0,MPI_COMM_WORLD);
7             MPI_Send(&matrix_size, 1, MPI_INT, i, 0,MPI_COMM_WORLD);
8         }
9         // ================ Declarar A_LOCAL ================ //
10        int rows, columns;
11        if(matrix_size%nprocs != 0){
12            rows = s+1;
13        }else{
14            rows = s;
15        }
16        columns = matrix_size;
17        double *A = (double *)calloc(rows*columns,sizeof(double));
18        // ================ Reparticion de MATRIX ================ //
19        int i_local = 0;
20        for (i=0; i<matrix_size; i++){
21            int dest = i%nprocs;
22            if (dest == root) {
23                memcpy(&A[i_local*columns], &matrix[i*columns], columns*sizeof(double));
24                i_local++;
25            } else {
26                MPI_Send(&(matrix[i*columns]), columns, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
27            }
28        }
29        time_end=MPI_Wtime();
30        time_sending_data = time_end-time_start;
```

CÓDIGO MPI:CÓDIGO PROCESADO MÁSTER CALCULO D

```
        // ================ Procesado ================ //
        double *D = (double *)calloc(1*columns,sizeof(double));
        double *v = (double *)calloc(1*columns,sizeof(double));
        time_start=MPI_Wtime();
        for (j = 0; j < matrix_size ; ++j) {
            if(pid == j%nprocs){
                int j_local = j/nprocs;
                for (i = 0; i < j ; ++i) {
                    v[i] = D[i]*A[i+j_local*columns];
                }

                double ts = 0;
                for (k = 0; k < j ; ++k) {
                    ts = ts + A[k+j_local*columns]*v[k];
                }
                v[j] = A[j+j_local*columns]-ts;
                A[j+j_local*columns] = v[j];
            }
            int sender = j%nprocs;

            if(matrix_size > j+1){
                MPI_Bcast(&(v[0]), columns, MPI_DOUBLE, sender, MPI_COMM_WORLD);
            }
            D[j] = v[j];
            ...
```

CÓDIGO MPI: CÓDIGO PROCESADO MÁSTER CALCULO L

```
1            ...
2            int start, end;
3            if(pid <= j%nprocs){
4                start = (j/nprocs)+1;
5            }else{
6                start = (j/nprocs);
7            }
8
9            if( (matrix_size%nprocs <= pid) && (matrix_size%nprocs != 0) ){
10               end = s-1;
11           }else{
12               end = s;
13           }
14
15           int i_local;
16           for (i_local = start; i_local < rows; ++i_local) {
17               double ts = 0;
18               for (k = 0; k < j ; ++k) {
19                   ts = ts + A[k+i_local*columns]*v[k];
20               }
21               A[j+i_local*columns] = (A[j+i_local*columns]-ts)/v[j];
22           }
23       }
24       // ================ Procesado ================ //
25       time_end=MPI_Wtime();
26       time_procesing = time_end - time_start;
27       ...
```

CÓDIGO MPI: CÓDIGO RECEPCIÓN DATOS

```
1         ...
2         // ================ Recepcion de Resultado ================ //
3         time_start=MPI_Wtime();
4         i_local = 0;
5         for (i=0; i<matrix_size; i++){
6             int origen = i%nprocs;
7             if (origen == root) {
8                 memcpy(&matrix[i*columns], &A[i_local*columns], columns*sizeof(double));
9                 i_local++;
10            } else {
11                MPI_Recv(&(matrix[i*columns]), columns, MPI_DOUBLE, origen, 0, MPI_COMM_WORLD,&status);
12            }
13        }
14
15        time_end=MPI_Wtime();
16        time_recovering_data = time_end-time_start;
17
18        printf("Frobenius Norm: %20.20f\n", norm(matrix, originalMatrix, matrix_size));
19
20        printf("\n*********************************\n\n");
21        printf("Selected :%s\n","MPI");
22        printf("Size of Matrix :%d \n",matrix_size);
23        printf("Sendig Matrix: %f seconds\n",time_sending_data);
24        printf("DECOMPOSE TIME TAKEN : %f seconds\n",time_procesing);
25        printf("Recovering result: %f seconds\n",time_recovering_data);
26        printf("Total Time Spent: %f seconds\n",time_sending_data+time_procesing+time_recovering_data);
27        printf("\n*********************************\n\n");
28
29        // ================ Recepcion de Resultado ================ //
```

CÓDIGO MPI: CÓDIGO OTROS RECEPCIÓN DATOS

```
1         int j,i,k;
2         int s, matrix_size;
3         // ================ Declarar A_LOCAL ================ //
4
5         MPI_Recv(&s, 1, MPI_INT, root, 0, MPI_COMM_WORLD, &status);
6         MPI_Recv(&matrix_size, 1, MPI_INT, root, 0, MPI_COMM_WORLD, &status);
7
8         int rows, columns;
9         if(matrix_size%nprocs != 0){
10            rows = s+1;
11        }else{
12            rows = s;
13        }
14        columns = matrix_size;
15
16        double *A = (double *)calloc(rows*columns,sizeof(double));
17        // ================ Reparticion de A_LOCAL ================ //
18
19        int size_transf;
20        if( (matrix_size%nprocs <= pid) ){
21            size_transf = s-1;
22        }else{
23            size_transf = s;
24        }
25
26        //printf("Size: %d\n",size_transf);
27
28        for (j=0; j<=size_transf; j++){
29            MPI_Recv(&A[j*columns], columns, MPI_DOUBLE, root, 0, MPI_COMM_WORLD,&status);
30        }
```

CÓDIGO MPI

Código Procesado:

```
Procesado D y L igual al máster.
```

Código Envío Datos Procesados:

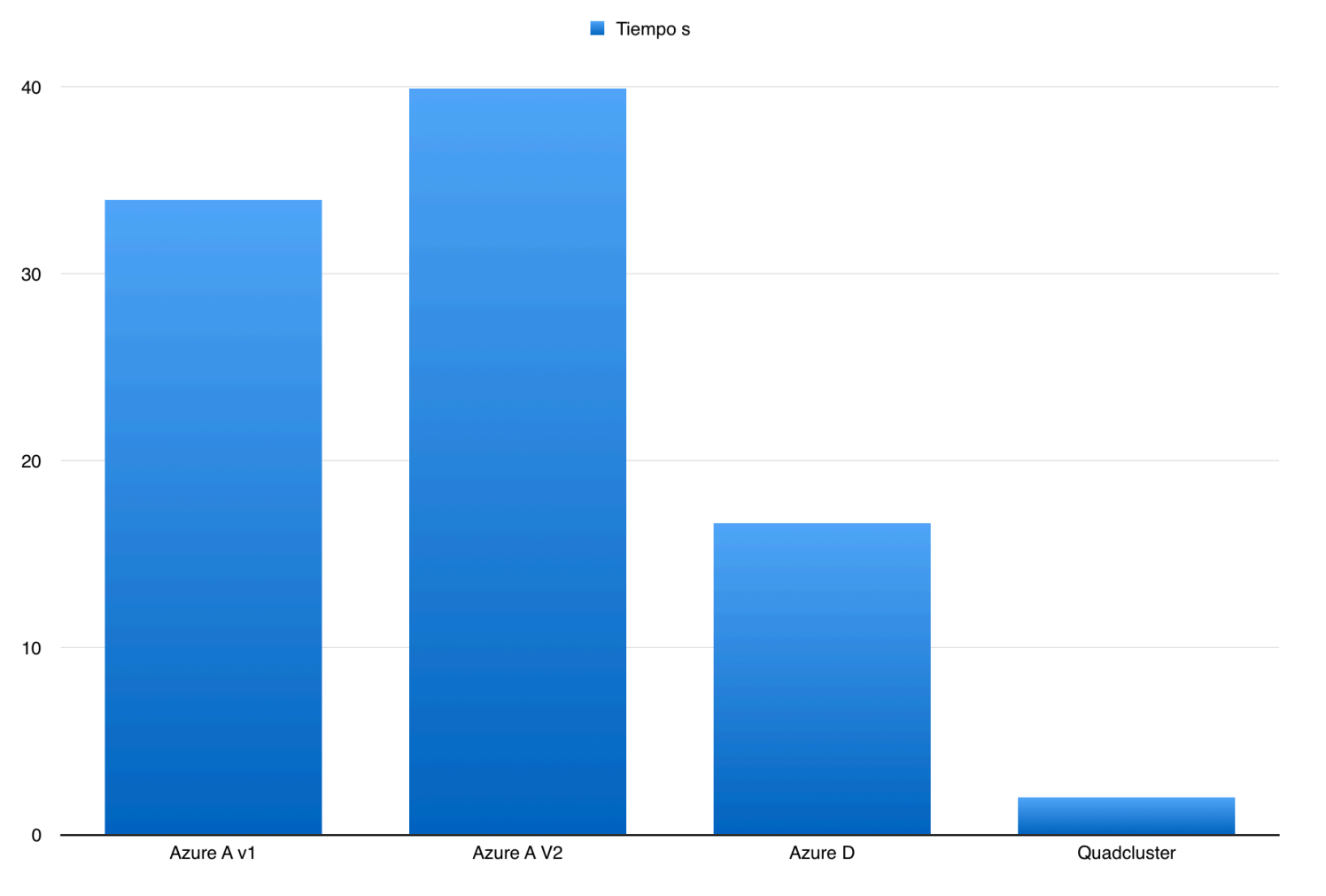
```
// ================ Envío de Resultado ================ //
for (j=0; j<=size_transf; j++){
    MPI_Send(&A[j*columns], columns, MPI_DOUBLE, root, 0, MPI_COMM_WORLD);
}
```

CÓDIGO MPI: EJECUCIÓN

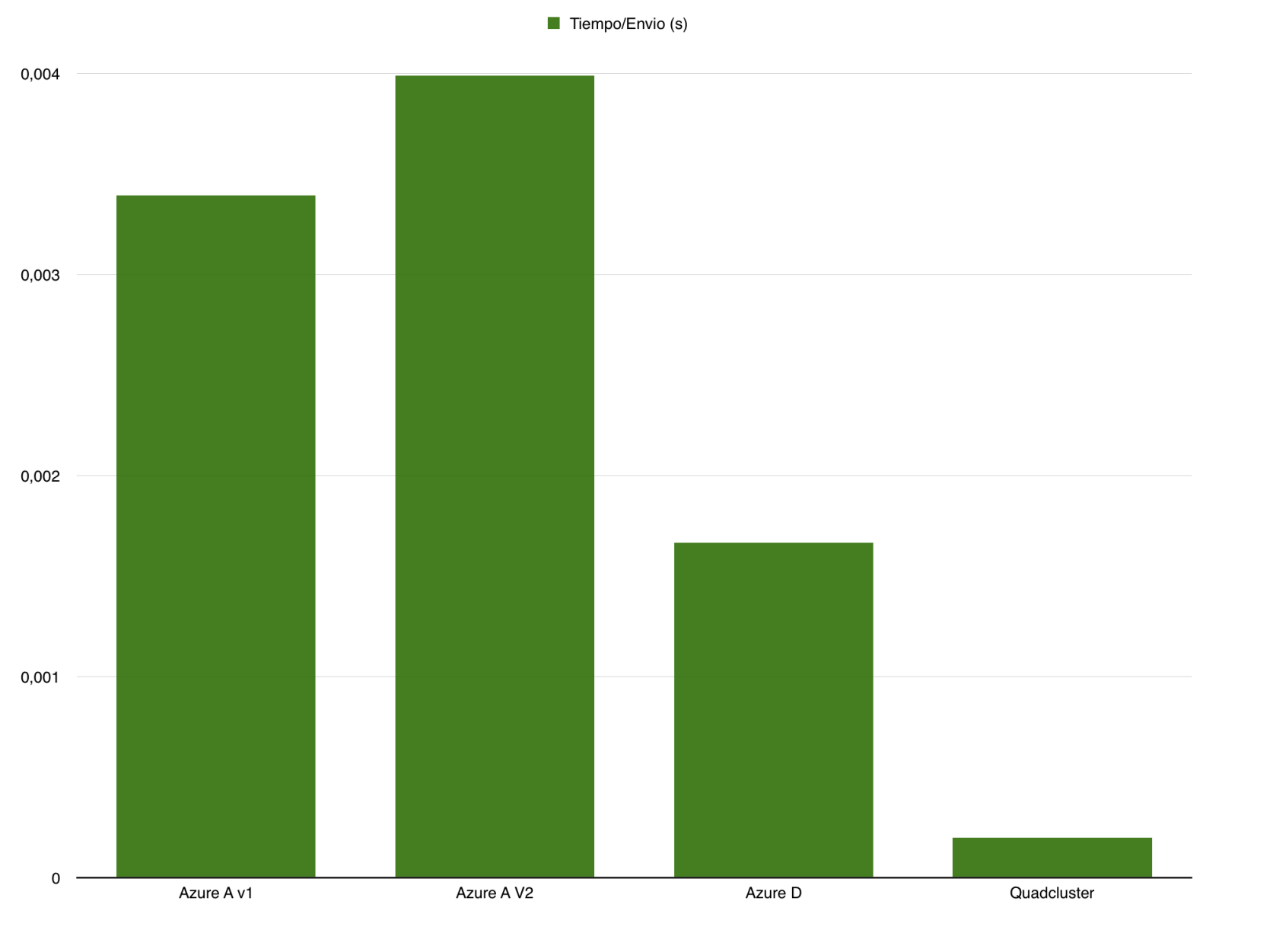
```
#!/usr/bin/env bash
if [ -z "$1" ]; then
    echo "First argument not supplied, compile.sh <size>(size>2) <cpuNumber>(cpuNumber>2) ";
    exit; fi
if [ -z "$2" ];then
    echo "Second argument not supplied, compile.sh <size>(size>2) <cpuNumber>(cpuNumber>2) ";
    exit;fi
echo "Creando machinefile"
# Se crea el machinefile automaticamente en base a los nodos que están conectados
echo "frontend" > machines
for (( i=0; i <= $2; i=i+1 ));do
    eval ping -c 3 "compute$i" >/dev/null
    if [ "$?" == "0" ]; then
        echo "compute$i" >> machines
    fi
done
make
echo "Enviando mpi.out"
for (( i=0; i <= $2; i=i+1 ))
do
    echo "Sending mpi.out to compute$i"
    scp mpi.out compute$i:.
done
echo "Ejecutando"
for (( j=1; j <= $2; j++ ));do
    for (( i=0; i <= $1; i=i+200 ));do
        echo $i
        mpirun -np $j -machinefile machines ./mpi.out $i >> tiempoMPI.txt
    done
done
make clean
```

RESULTADOS

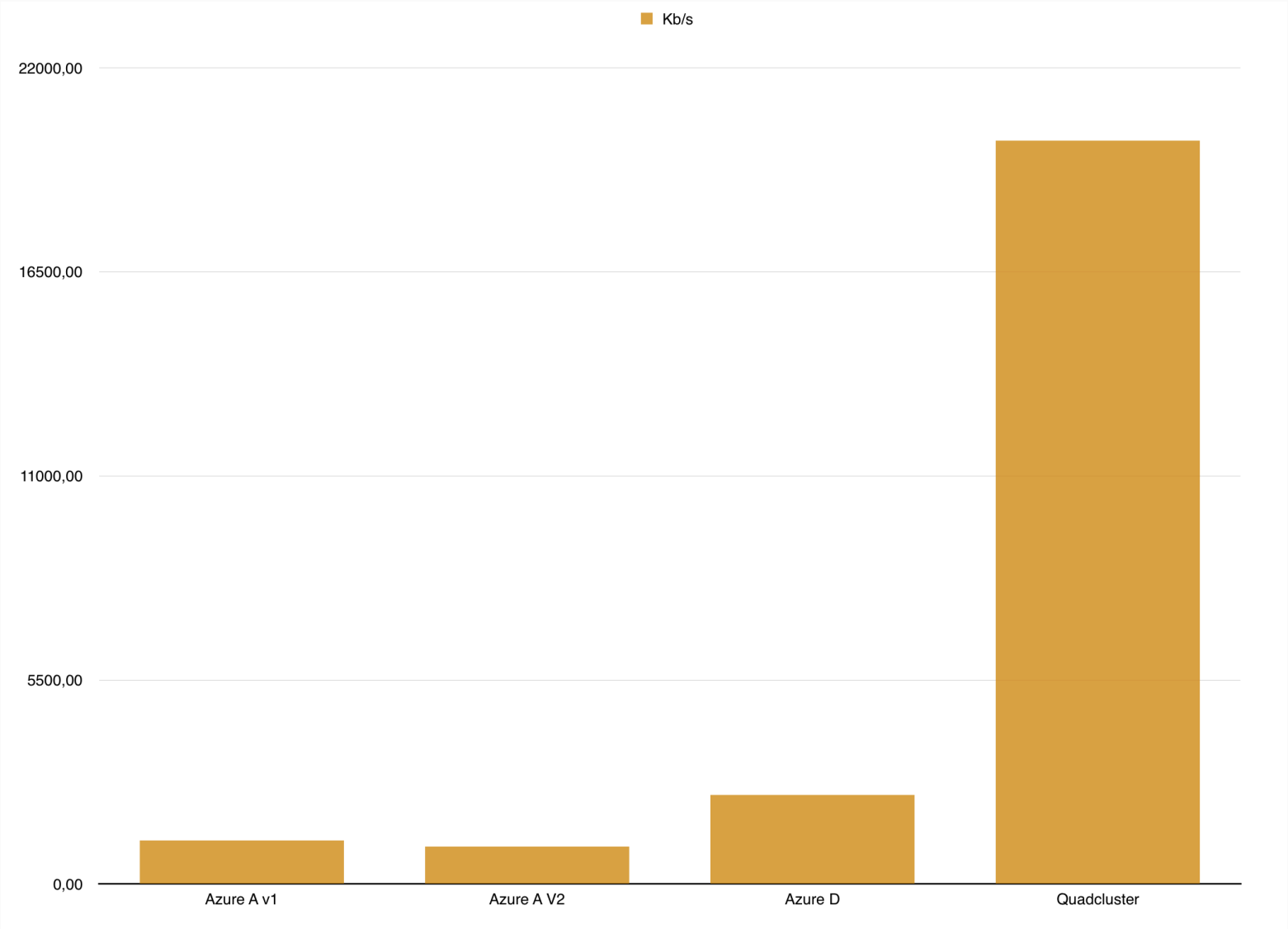
RESULTADOS: TIEMPO EMPLEADO 4 KB



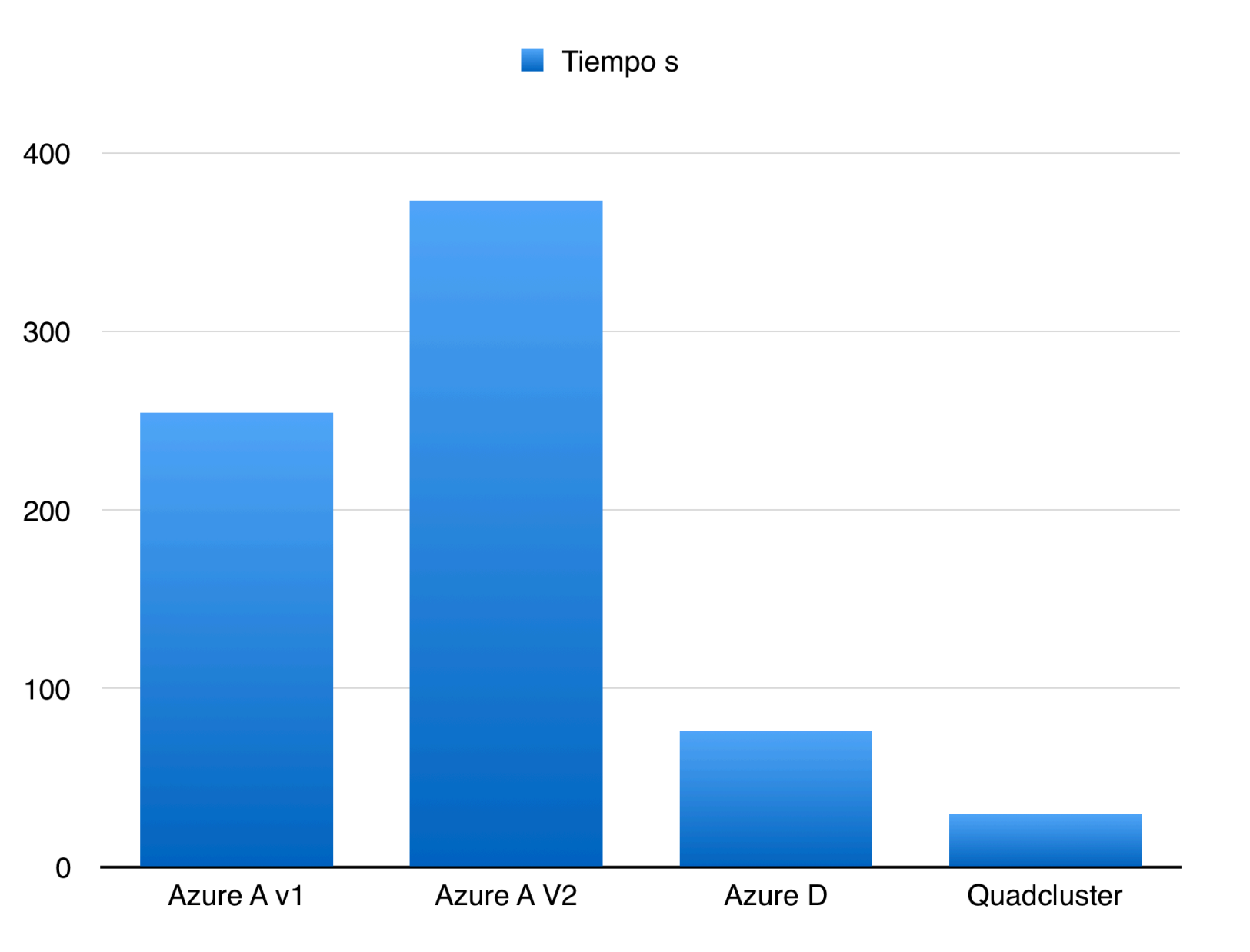
RESULTADOS: TIEMPO/ENVIO (S) 4 KB



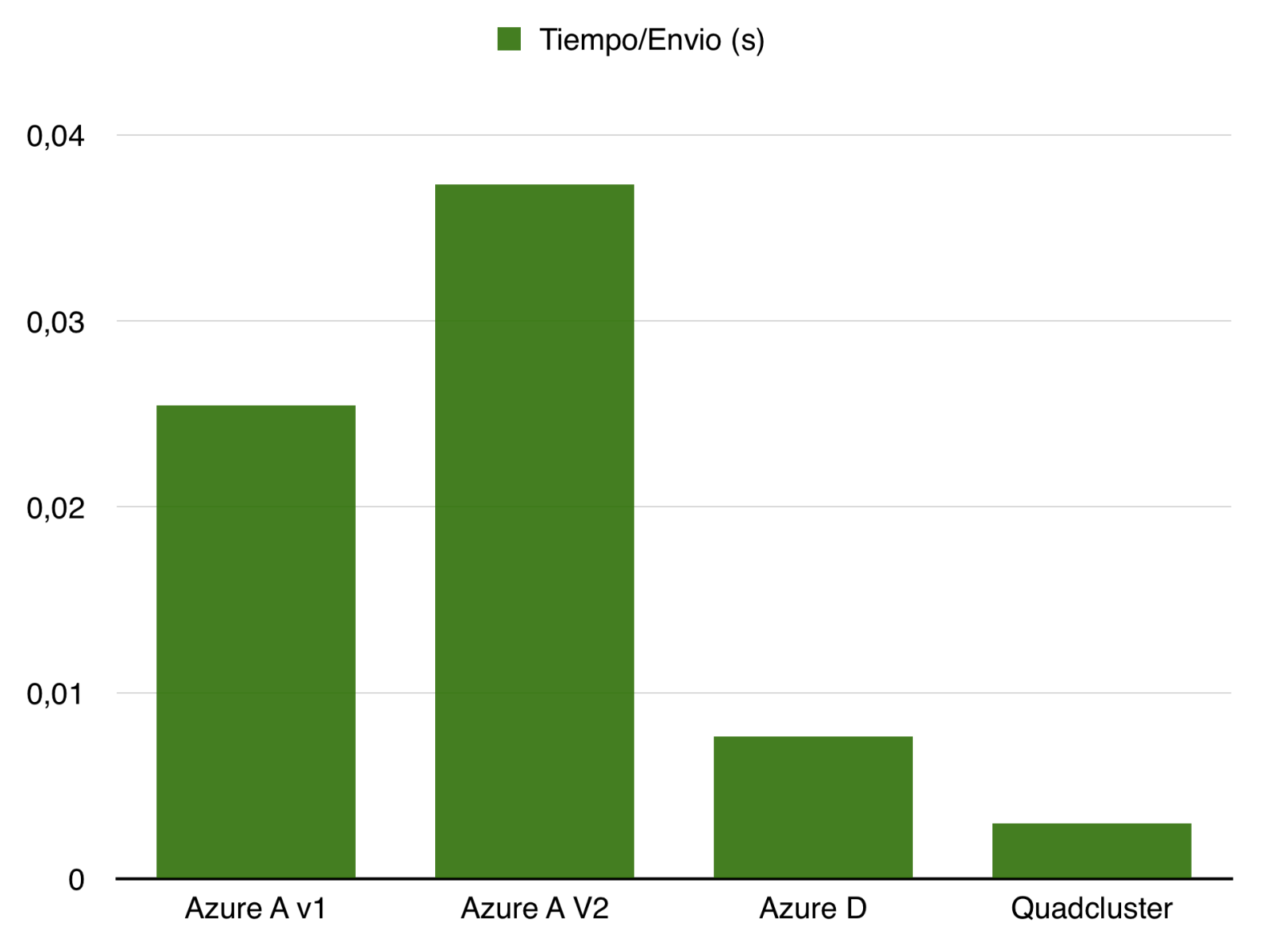
RESULTADOS: KB/S 4 KB



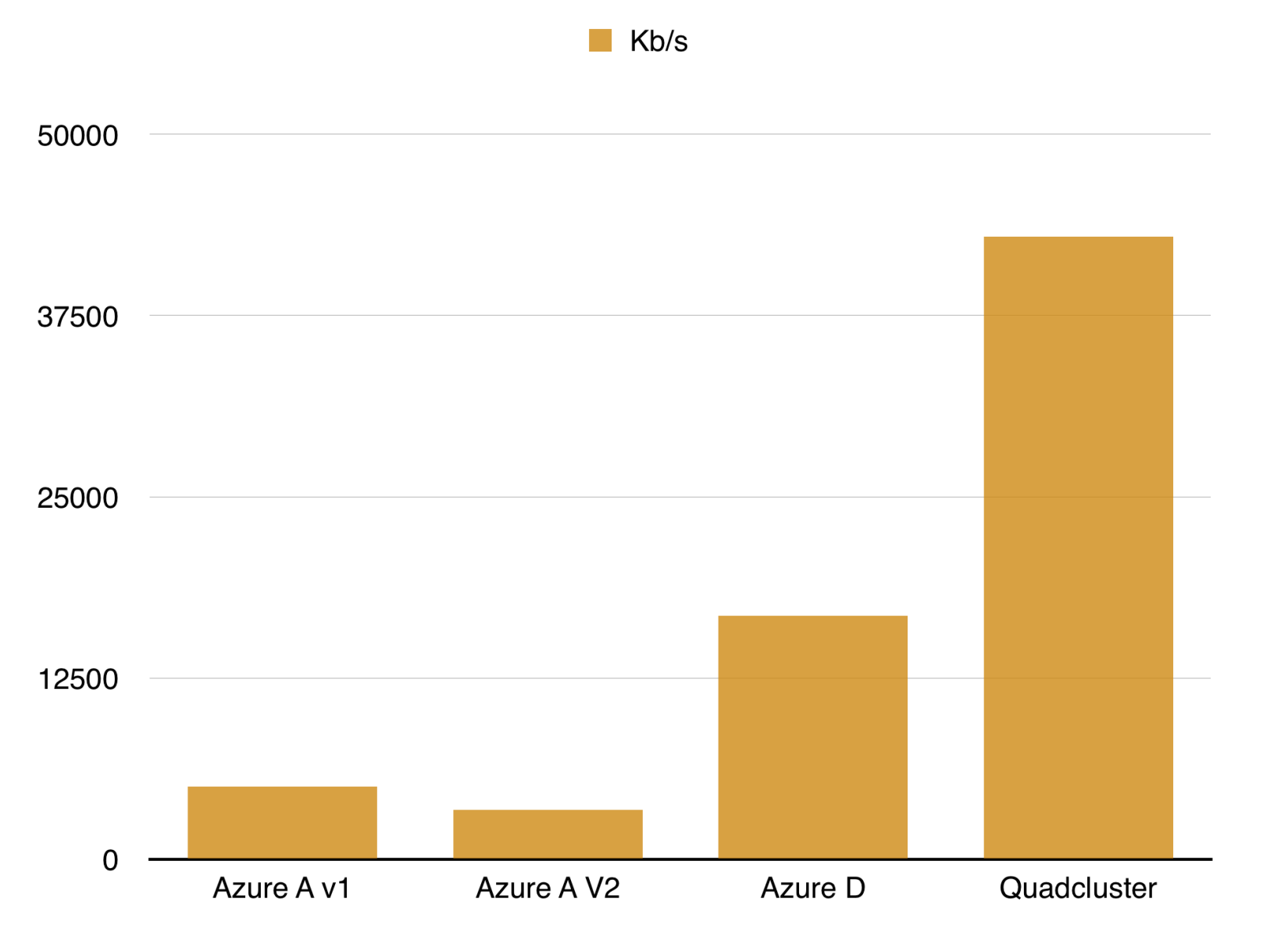
RESULTADOS: TIEMPO EMPLEADO 128 KB



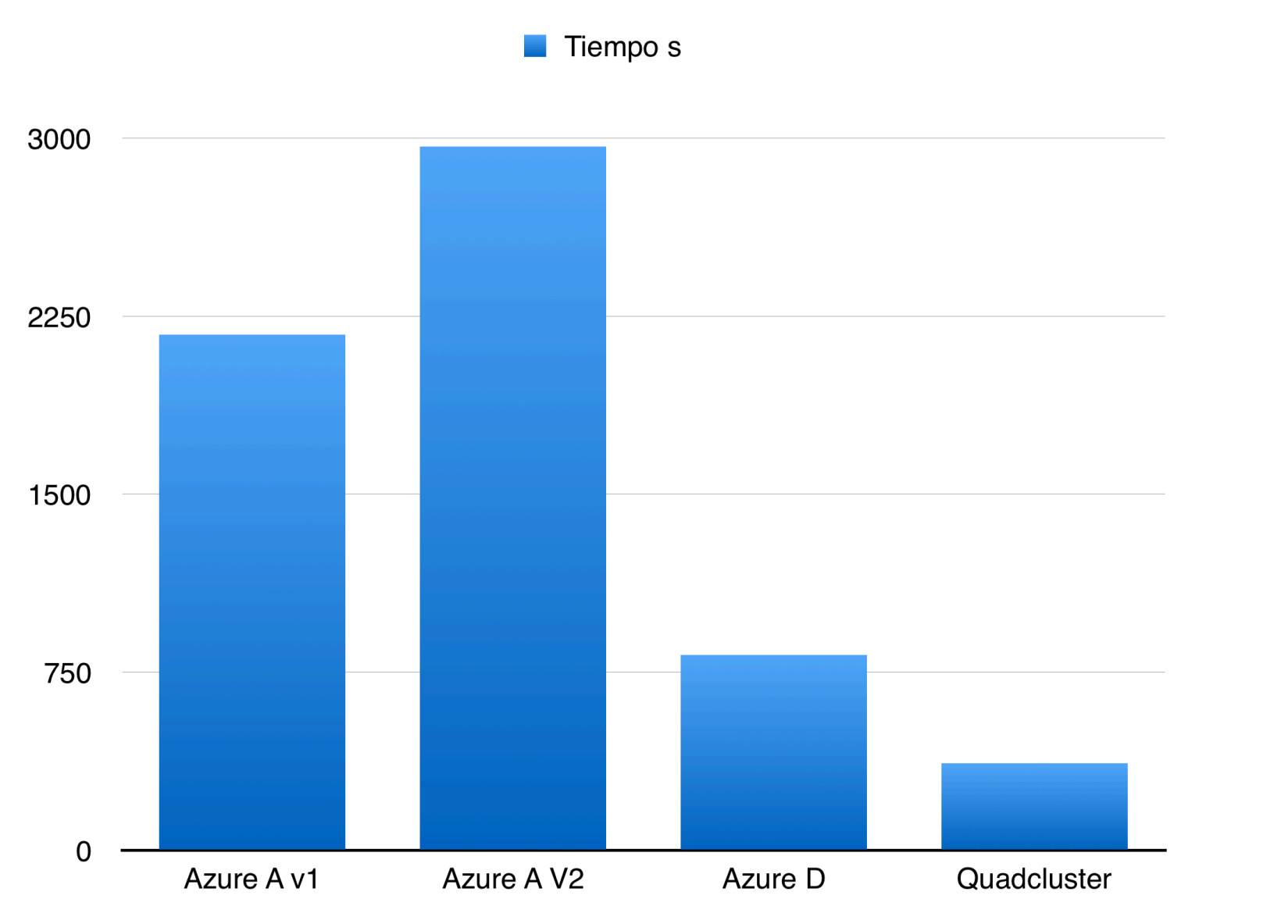
RESULTADOS: TIEMPO/ENVIO (S) 128KB



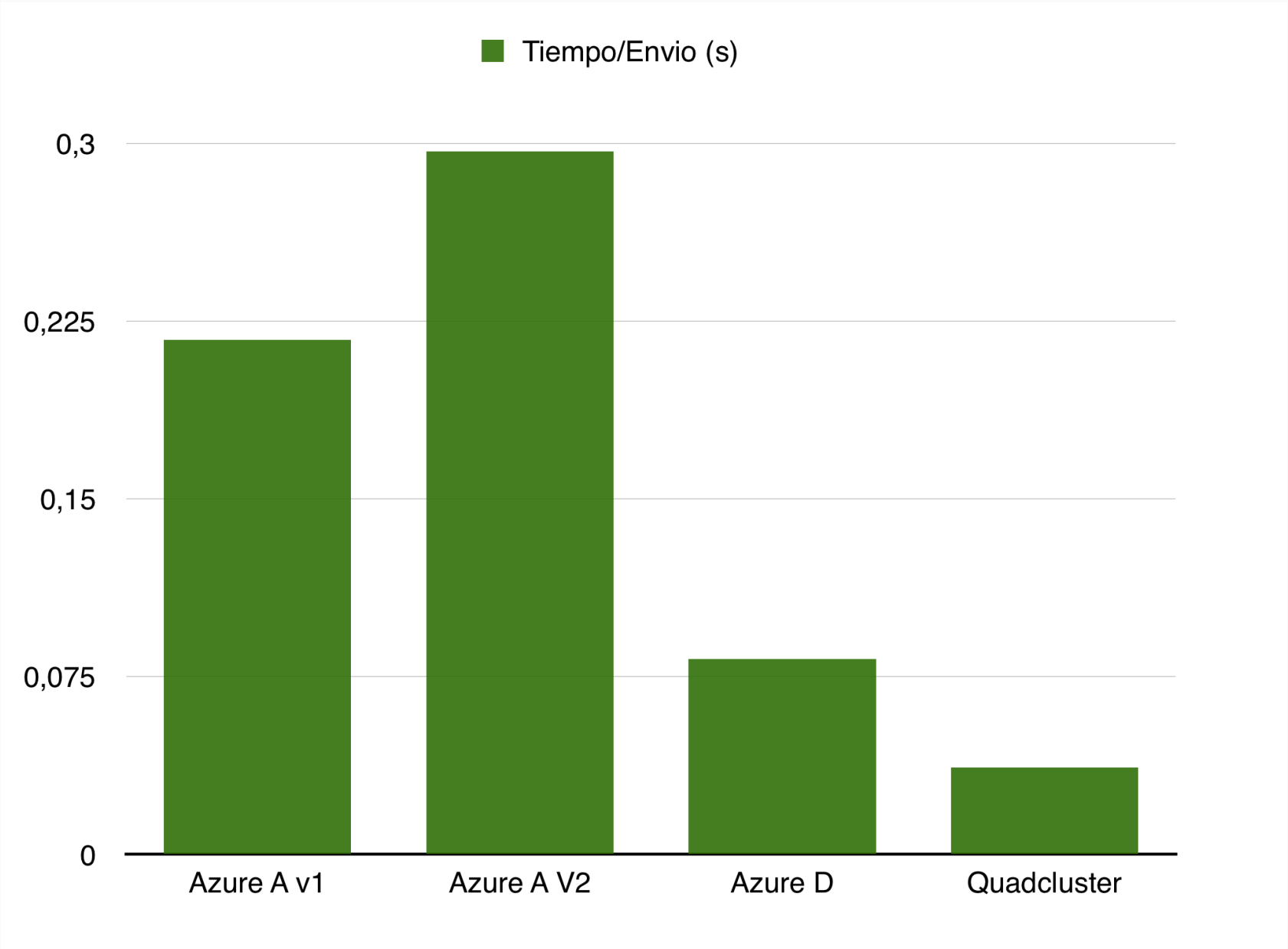
RESULTADOS: KB/S 128KB



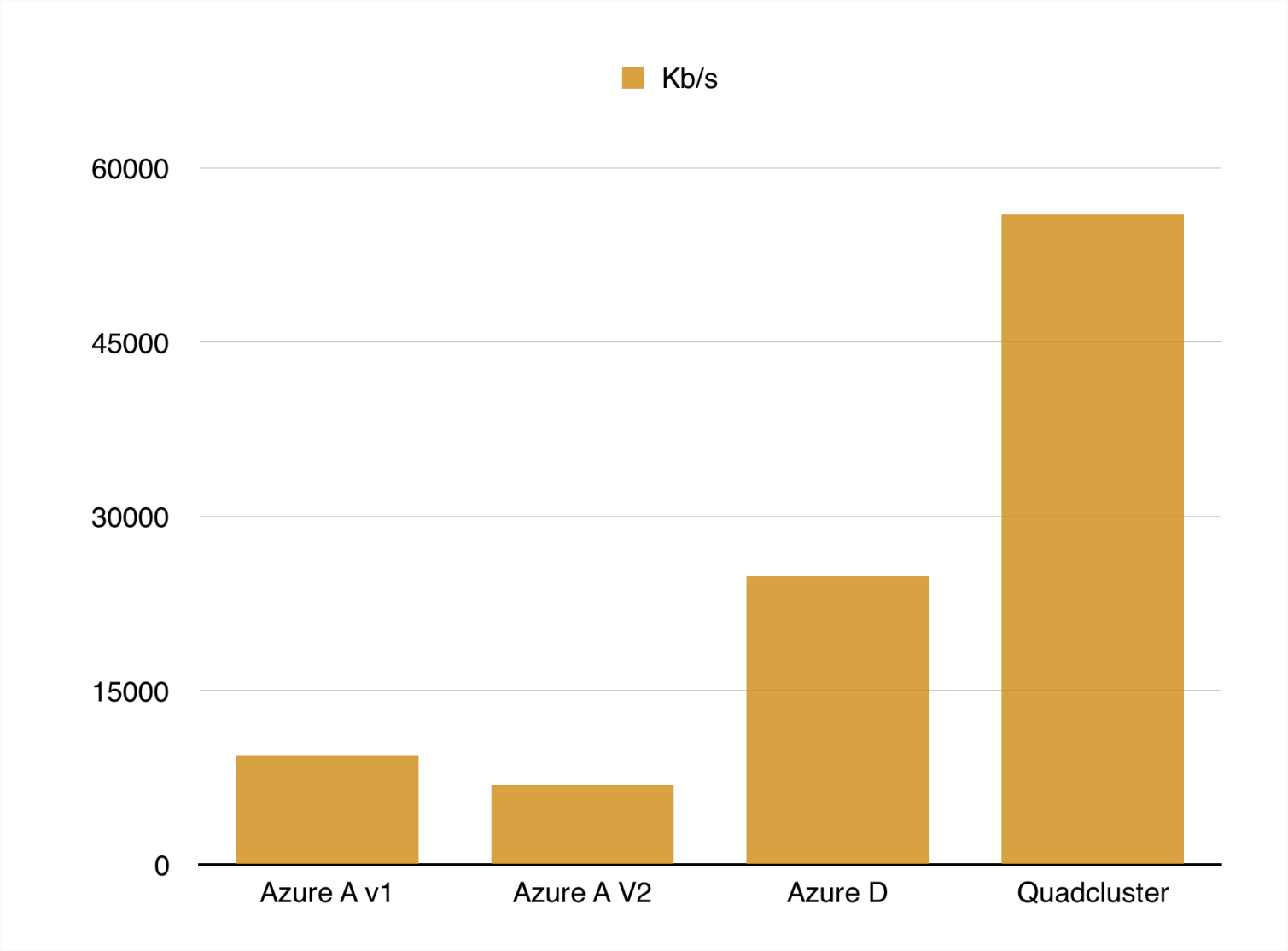
RESULTADOS: TIEMPO EMPLEADO 2048 KB



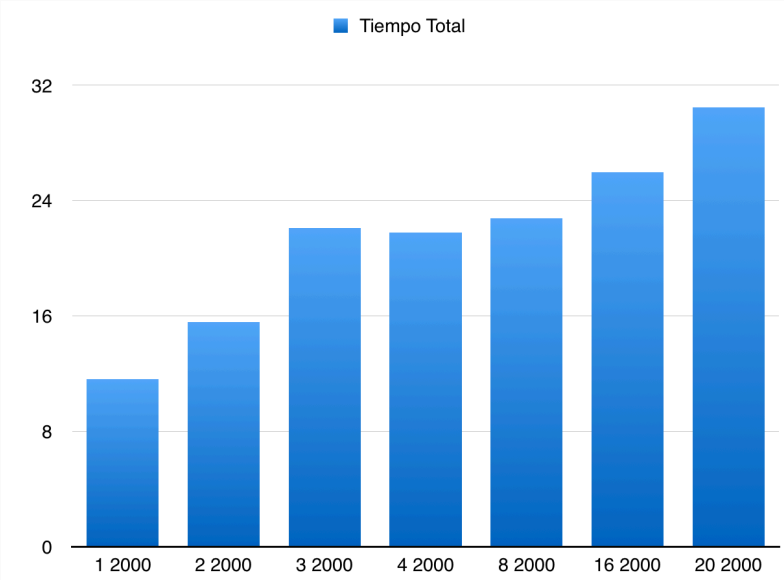
RESULTADOS: TIEMPO/ENVIO (S) 2048 KB

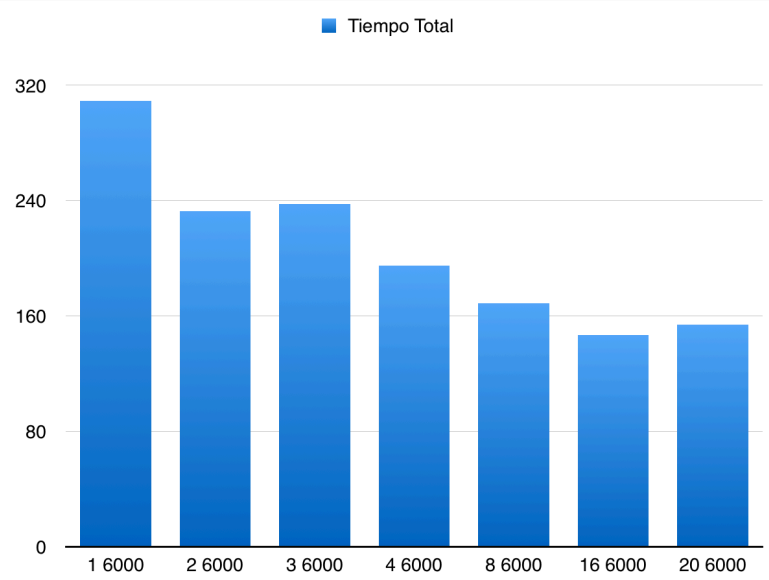


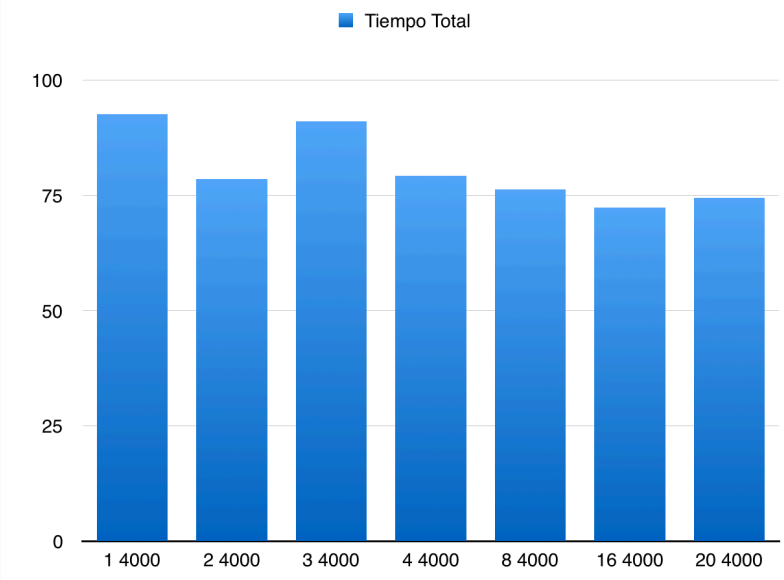
RESULTADOS: KB/S 2048 KB

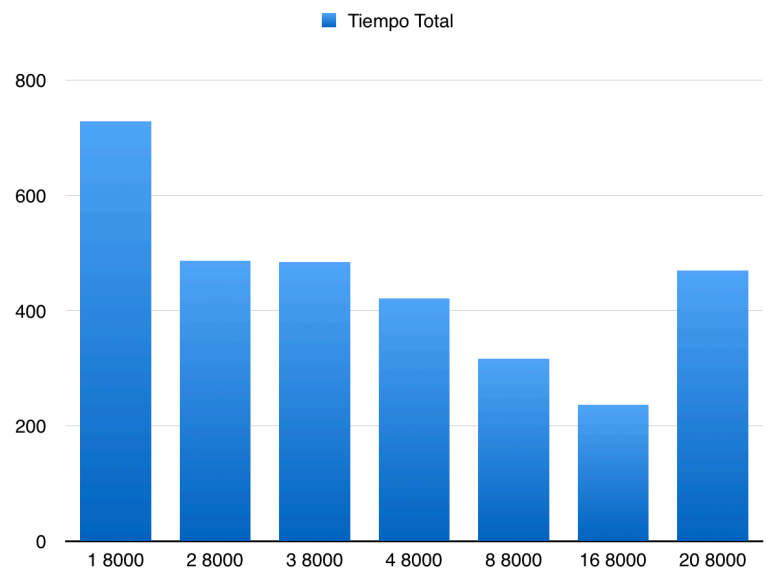


RESULTADOS: AZURE V1 TIEMPOS

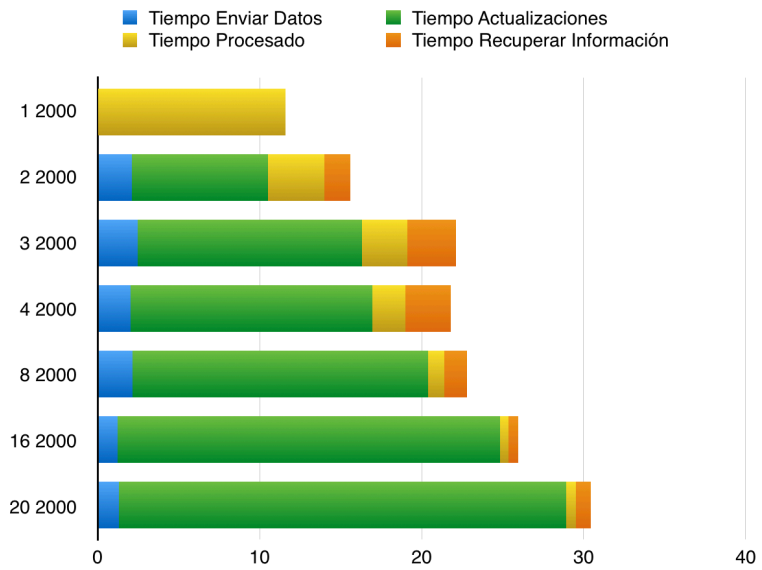


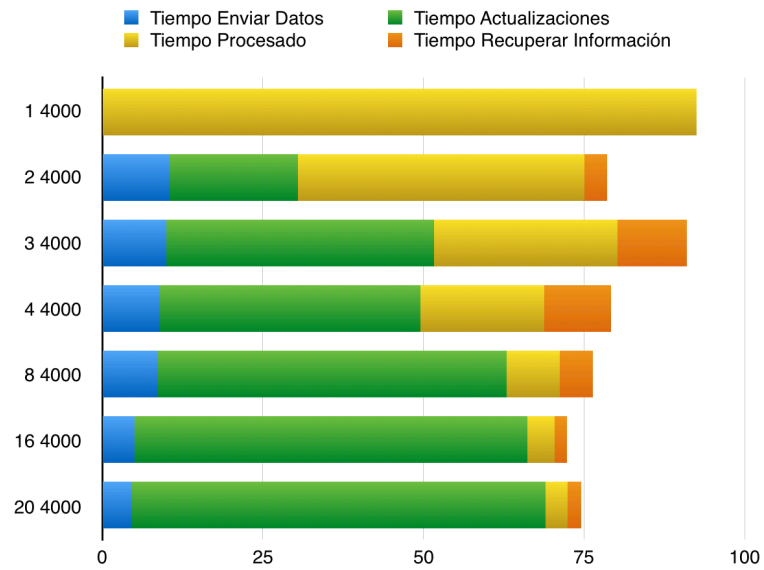


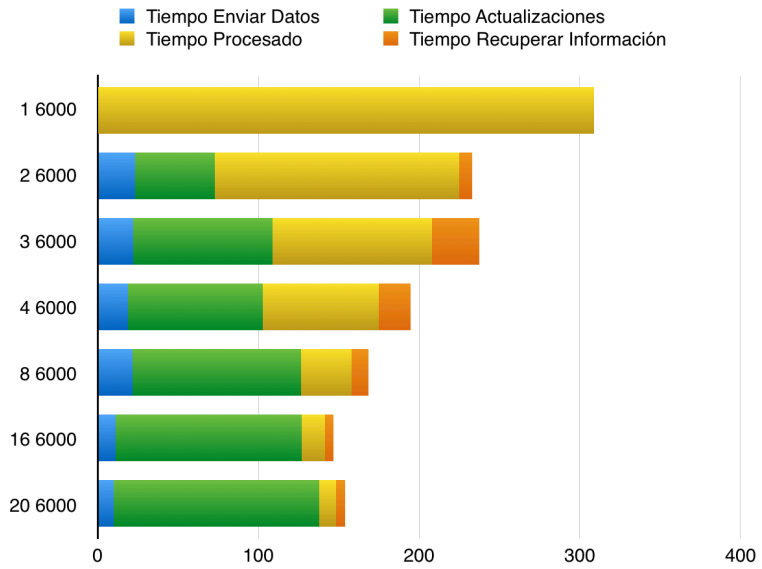


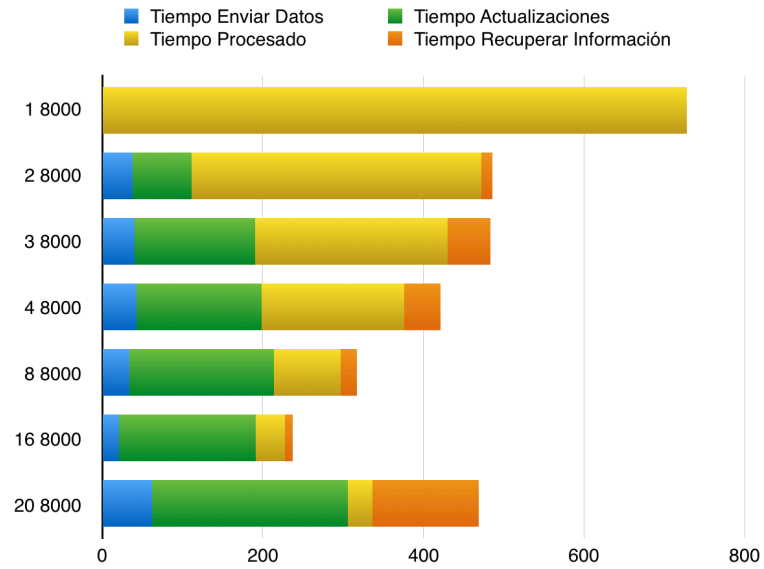


RESULTADOS: AZURE V1 DESGLOSE

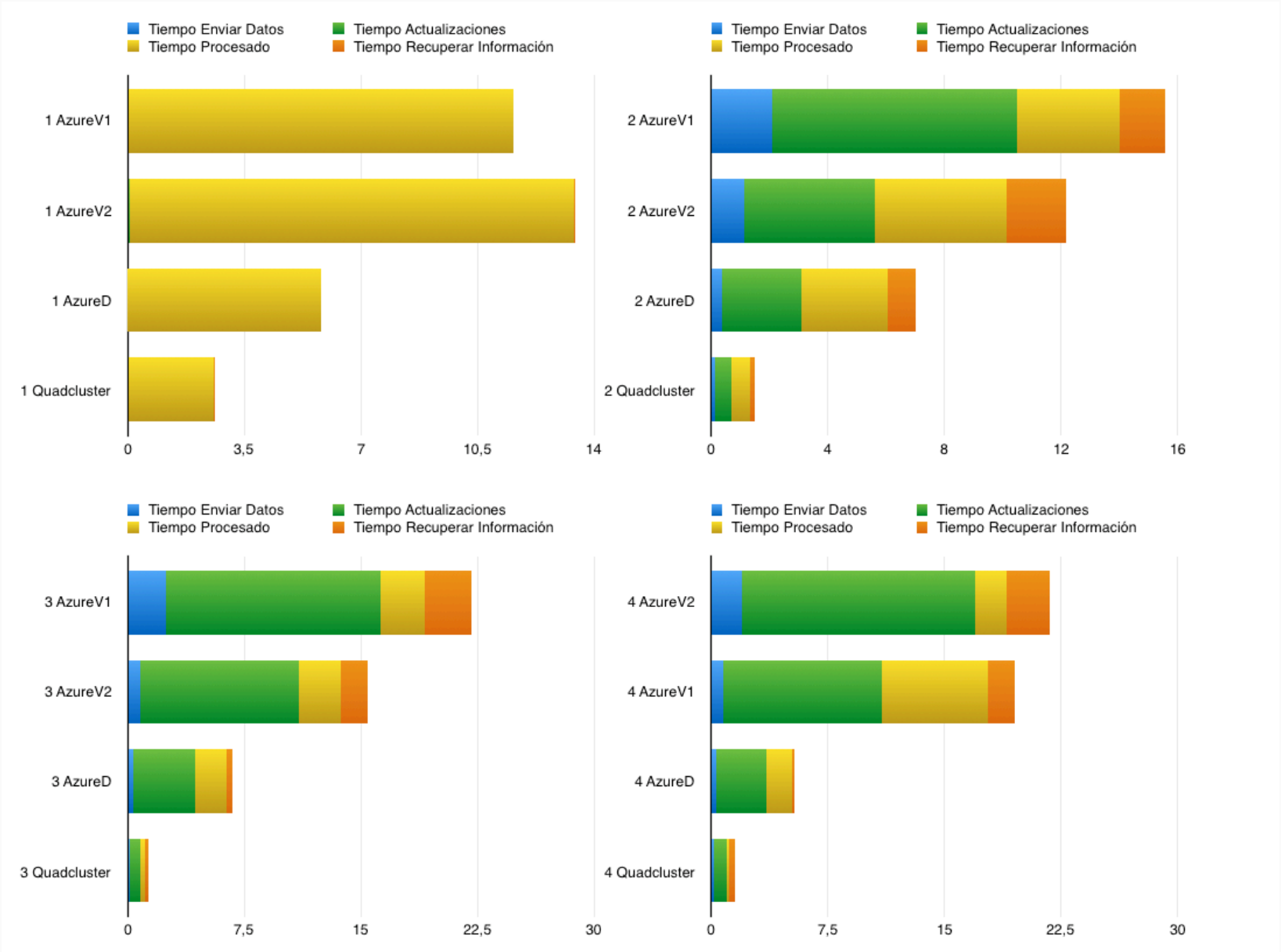




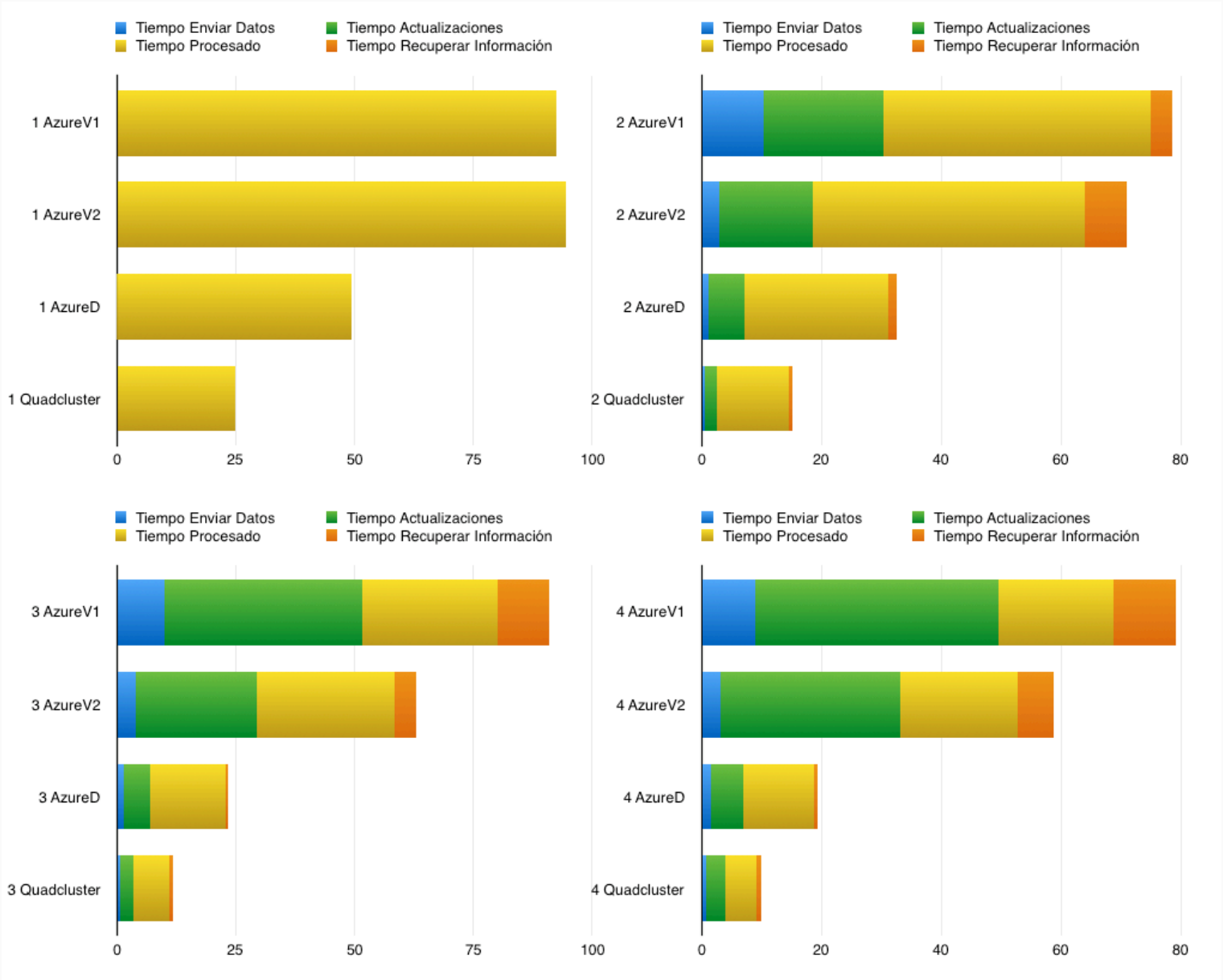




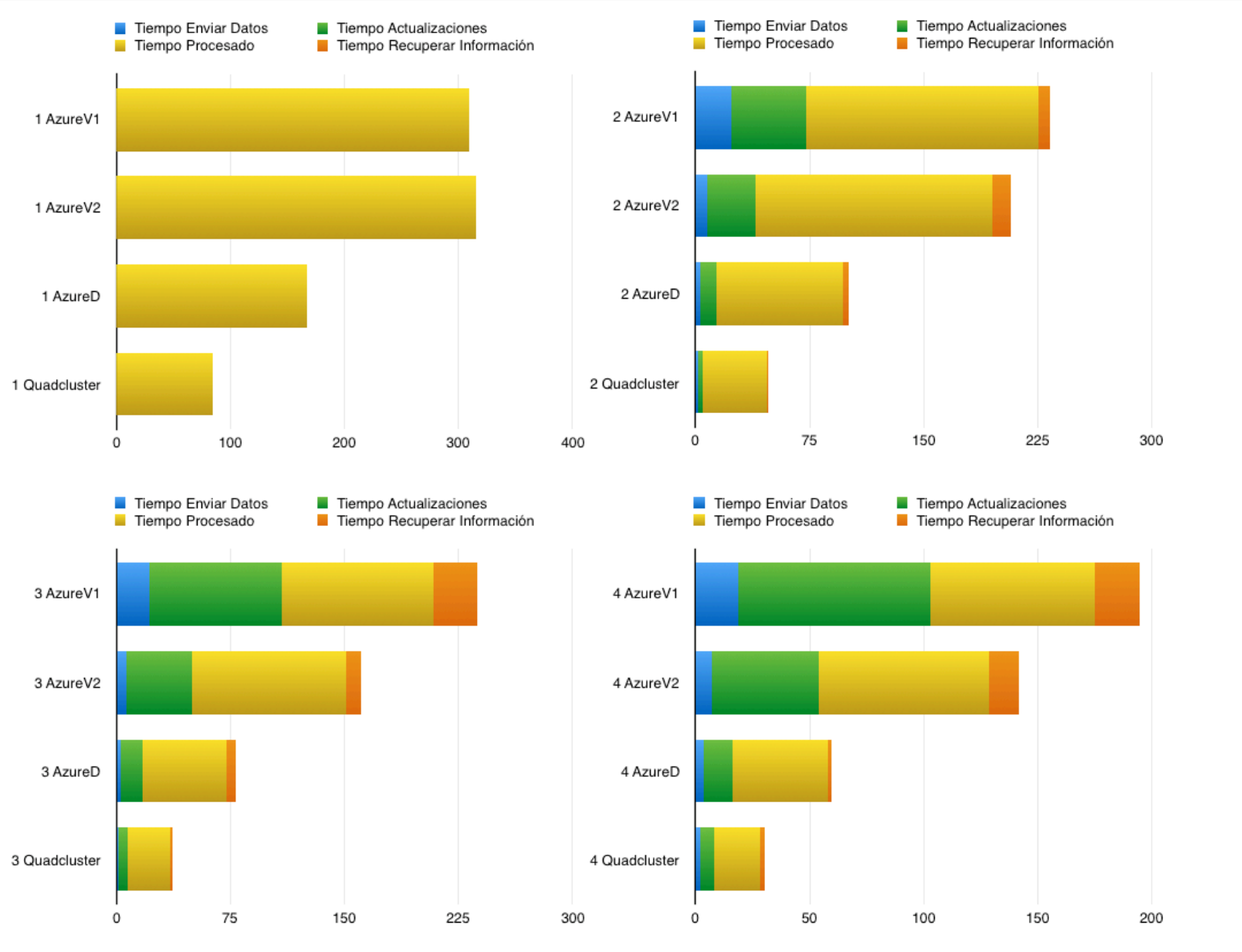
RESULTADOS: AZURE MATRIZ 2000X2000



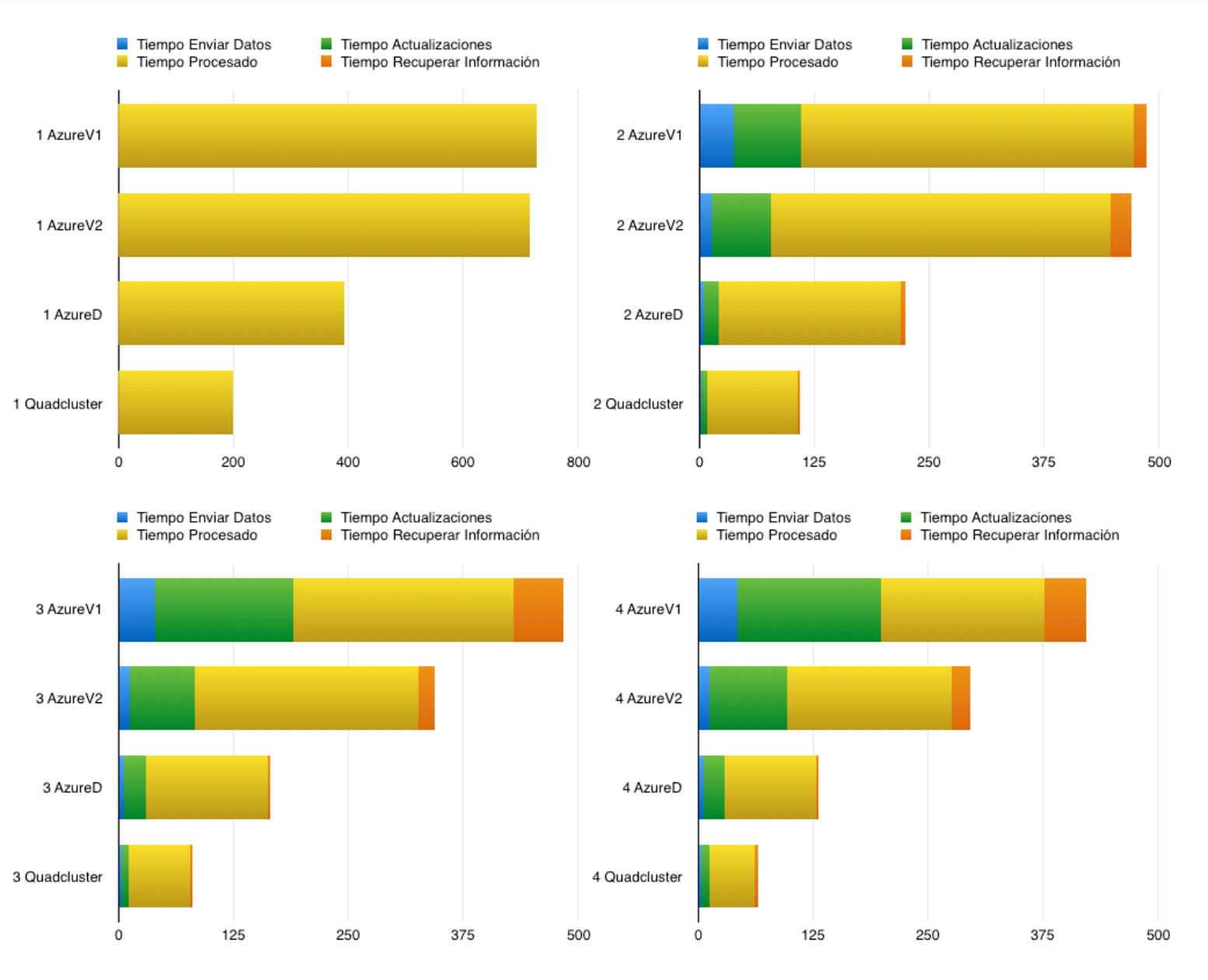
RESULTADOS: AZURE MATRIZ 4000X4000



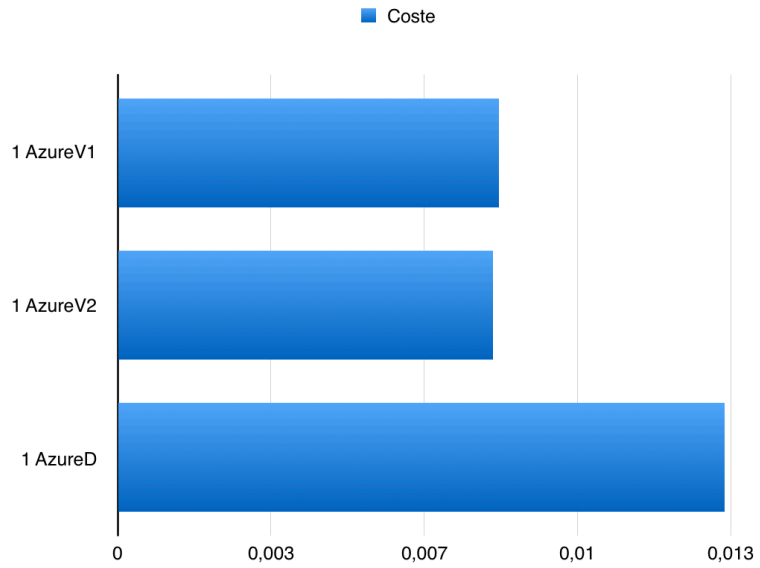
RESULTADOS: AZURE MATRIZ 6000X6000

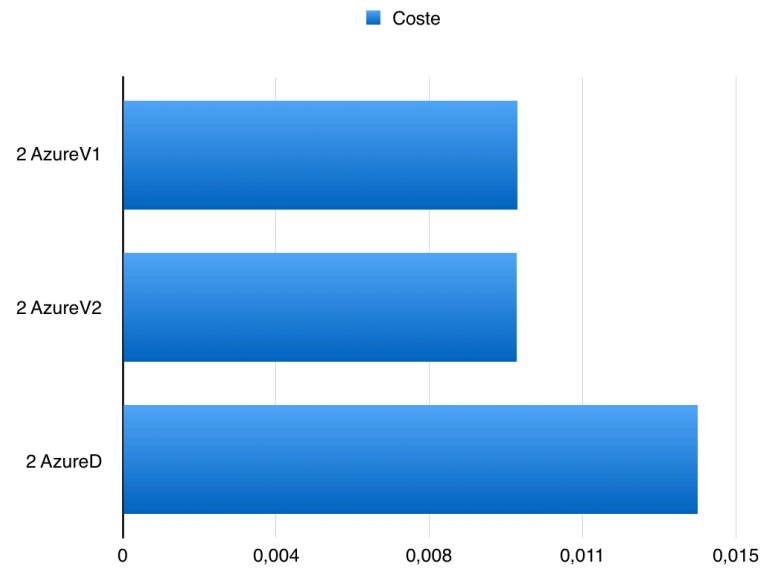


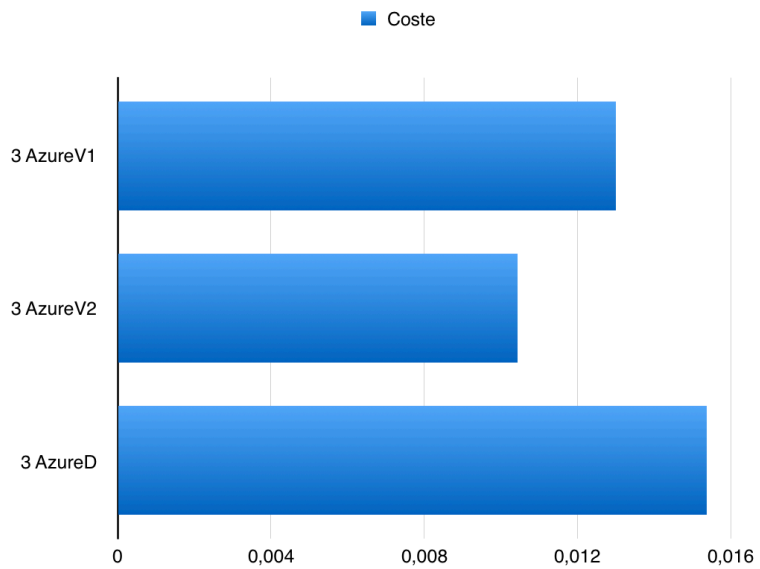
RESULTADOS: AZURE MATRIZ 8000X8000

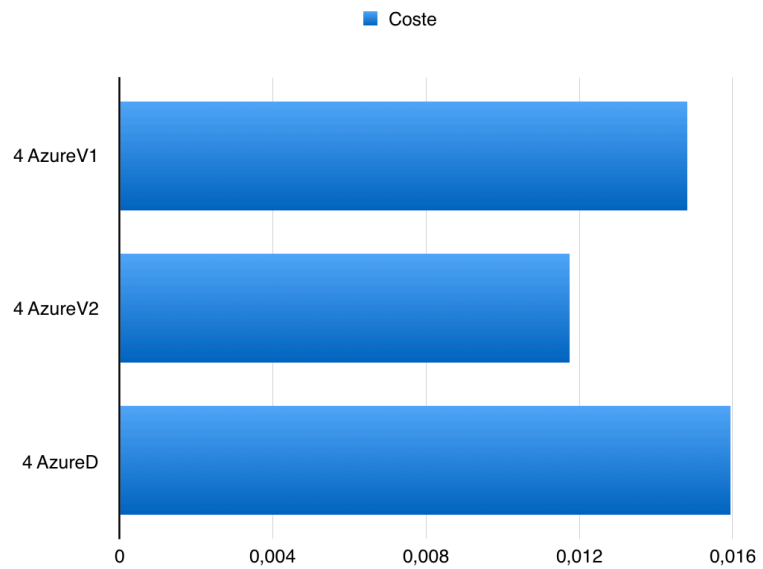


RESULTADOS: COSTES









CONCLUSION Y TRABAJOS FUTUROS

CONCLUSION Y TRABAJOS FUTUROS

El algoritmo de la factorización LDL^T implementar de manera concurrente y con unas mejoras considerables.

Se ha podido observar que el claud computing de Azure puede ser una buena alternativa y mucho más barata para cálculos pesados si no se dispone de un equipo adecuado.

Para futuras mejoras se debería de utilizar BLASH para la actualización de la matriz en MPI y utilizar un algoritmo por bloques para reducir las comunicaciones.

Se debería de intentar realizar pruebas con más de 4 nodos de tipo D1 para ver cuando se puede conseguir de mejor, además se deberían de probar otros proveedores de cloud computing como Amazon para comparar sus servicios.

RESUMEN

Pueden conseguir todo el código fuente en

`github.com/MihaiLupoiu/CMCP_LDL_Factorization`

¡Gracias!

¿Alguna pregunta?